

Technical Report B004

DTIC FILE COPY

(2)

AD-A218 694

Technical Report (Final)
Issued May 31, 1988
Duration October 1, 1987 to March 31, 1988

Prepared Under
Contract Number DAAB07-87-C-A035

DTIC
ELECTRIC
MAR 02 1990
S D G D

PHASE I FINAL REPORT

Final Report of
Army SBIR Phase I Project
Advanced Facilities to Expedite Design and
Evaluation of Communications Systems

Prepared For

Frank Giordano, Project Leader
US Army CECOM
Fort Monmouth, NJ 07703

by

Information Research Associates
911 West 29th Street
Austin, Texas 78705



Doug M. Neuse
Principal Investigator

The view, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless designated by other documentation.

DTIC
Approved for Release
Distribution Unlimited

20 03 01 055

CECOM COMMUNICATIONS MODELING PROJECT PHASE I FINAL REPORT

TABLE OF CONTENTS

Abstract.....	1
1.0 Introduction	2
2.0 Problem Statement.....	4
3.0 Research Approach.....	6
3.1 Requirements and Experimentation Specification.....	6
3.2 Model Implementation and Experimentation.....	6
3.3 Enhancement Specification.....	7
3.4 Training.....	8
4.0 Previous Related Work.....	9
5.0 Modeling and Experimental Evaluation Results.....	11
5.1 Data Link Control Protocol.....	11
5.2 Switching Network Model	13
5.3 Communications Switch Model	16
6.0 Summary of Enhancement Specification.....	19
7.0 Conclusions.....	21
8.0 References.....	22
Appendix A: Modeling and Experimental Evaluation Report.....	A-1
Appendix B: C/PAWS Enhancement Specification.....	B-1
Appendix C: Outline of PAWS and GPSM.....	C-1



Accession No.	1
NTIS	<input checked="" type="checkbox"/>
DTIC	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	<i>per ltr.</i>
Date	
Dist	
A-1	

ABSTRACT

This report describes the technical accomplishments and results of the Army CECOM sponsored SBIR Project "Advanced Facilities to Expedite Design and Evaluation of Communications Systems". The goal of this project was to determine the feasibility of an approach to high level simulation modeling of communications system. The approach is to adapt ES/PAWS, a top-down design tool for electronic systems being developed for the Navy [NAV 87, BRO 88]. The ES/PAWS tool is based on existing machine simulation products developed by Information Research Associates (IRA). These products, PAWS and GPSM [IRA 87a, IRA 87b], offer high-level visual and declarative facilities for constructing simulation models.

We have 1) evaluated the requirements for a powerful high-level modeling tool for communications systems; 2) constructed several experimental models to gain further insight into the issues arising in communications system modeling, and drawn upon IRA's substantial previous work in this area and 3) specified the enhancements required to ES/PAWS in order to construct an effective high-level communication modeling tool, to be called C/PAWS.

We have established that the ES/PAWS tool currently under development will provide many powerful features useful for communications modeling. As part of the ES/PAWS development the graphical interface, GPSM, will also be substantially enhanced. We have further specified that C/PAWS will contain many advanced high-level features for communications modeling, including provision for libraries of re-usable submodels and C functions, for modeling common communication components and algorithms. The C/PAWS tool will also provide user-definable icons, improved graphical output of simulation results, and specialized support for modeling communication features such as timeouts and component failures.

The strategy of adapting ongoing development work and specializing it for communications system modeling has been established as a very efficient and cost-effective approach to developing C/PAWS. Based on the promising finding of these preliminary studies, we recommend the development of a prototype system for advanced modeling and evaluation of communications systems.

1.0 Introduction

This report describes the technical accomplishments and results of the Army CECOM sponsored SBIR Project "Advanced Facilities to Expedite Design and Evaluation of Communications Systems". The goal of this project was to determine the feasibility of an approach to high level simulation modeling of communications systems. The approach is to adapt ES/PAWS, a top-down design tool for electronic systems being developed for the Navy [NAV 87, BRO 88]. The ES/PAWS tool is based on existing mature simulation products developed by Information Research Associates (IRA). These products, PAWS and GPSM [IRA 87a, IRA 87b], offer high-level visual and declarative facilities for constructing simulation models.

ES/PAWS is to be a major new product offering from IRA that will build upon the highly effective modeling facilities of PAWS and GPSM. It will provide the expressiveness of the C language, support for macro pre-processing and finite state machine specification, submodel parameterization, and an enhanced version of GPSM which includes a graph library facility, improved on-line help, and a structured input facility. The simulation tool proposed for the Army, called C/PAWS, will enhance ES/PAWS to provide even more powerful facilities for modeling communication systems.

The problem statement and the research approach are described in sections 2.0, and 3.0. There has been a substantial amount of previous work in this area by IRA and IRA clients. This experience has been used in this project, particularly in the area of modeling the effect of failures in distributed systems, as described in section 4.0.

The project accomplished several tasks, the first being the evaluation of the requirements for a powerful high-level modeling tool for Army communication systems [IRA 88]. The succeeding tasks included simulation modeling and experimental evaluation of some representative Army communications system, and specification of the enhancements required to ES/PAWS in order to make modeling of communications systems even easier and more efficient.

The object of the modeling and experimental evaluation task of this project was to construct prototype models of some aspects of Army communications systems. These include models of communications components, protocol and algorithms from several different layers of the communications systems and included the modeling of the effect of failures on

the system. The principal results of this task are summarized in section 5.0, and the report describing the modeling experimental evaluation task is included as an attachment to this report.

The enhancement specification task discussed the features and enhancements required to ES/PAWS in order to make C/PAWS an elegant high-level simulation tool for Army communications systems. Since the actual implementation of C/PAWS is to be done in a Phase II project, the specification discussed the functional enhancements required, rather than specifying the implementation strategy. The principal results of this task are summarized in section 6.0, and the C/PAWS enhancement specification is included as an attachment to this report.

The previous work done by IRA in this area, the experimental evaluation, and the enhancements specified for ES/PAWS have established the feasibility of using C/PAWS to model the Army communications systems. Adapting the ongoing ES/PAWS development work for C/PAWS is a particularly efficient and cost-effective approach to building a specialized communications modeling tool. Based upon the promising findings of these preliminary studies, we recommend the development of a prototype system for advanced modeling and evaluation of communications systems.

2.0 Problem Statement

The general problem area addressed by this project was the modeling of communications networks of interest to the United States Army. The specific problem was:

- a) to assess the feasibility of using a prototype modeling tool, ES/PAWS, to satisfy the requirements of the U.S. Army for rapid and effective modeling of such networks, and
- b) to determine any enhancements to this tool necessary for such modeling.

ES/PAWS is a tool for top-down design and analysis of electronics systems. It is being developed under a Phase II SBIR grant from the United States Navy, and is based upon the Performance Analyst's Workbench System (PAWS) [IRA 87a] and Graphical Programming of Simulation Models (GPSM) [IRA 87b].

PAWS is a successful, commercially available simulation language. It provides a high-level declarative specification language that reduces modeling errors and speeds model development compared to traditional procedural simulation languages. GPSM is the graphical front end to PAWS. It allows the user to develop PAWS models by drawing pictures (the pictures are the models), thereby reducing modeling errors and speeding model development further. GPSM also provides excellent presentation materials (pictures) for these models (see appendix C).

The modeling of U.S. Army communications networks has unusual requirements.

- a) The performance of commercial systems is primarily concerned with optimality under rather stable operating conditions. The performance of military systems is primarily concerned with adequacy under dynamic conditions.
- b) Performance in the presence of faults and performance with degraded resources are of prime importance.
- c) Since so many communications protocols and routing algorithms are in use, an effective communications modeling tool requires a library of submodels for

such protocols and algorithms, with a capability for user modification of this library.

- d) The network requirements change rapidly. Protocols and routing algorithms must be designed and analyzed rapidly. Models of these protocols and routing algorithms must therefore support correctness validation as well as performance analysis.

3.0 Research Approach

The approach we took to this problem is described below.

3.1 Requirements and Experimentation Specification.

The first task in this project was to specify the requirements for the graphical interface, simulation system and submodel library, and to define a plan for experimental demonstration of the modeling system. IRA representatives visited CECOM early in this project. We interviewed representatives of CECOM and obtained documents describing Army communications systems and modeling procedures. We studied the documents and professional literature in this area, and discussed our ideas with CECOM staff and with Professor Simon Lam, a noted expert in analysis of communication systems.

Base upon these interviews, analysis of the documents and literature, and discussions with CECOM staff and Professor Lam, we prepared a requirements and experimentation specification report [IRA 88].

The requirements included the following:

- a) suitability for use by communications engineers as well as network designers and analysts, and
- b) ability to model and evaluate a variety of networks including:
 - circuit-switched networks,
 - TDMA networks,
 - packet-switched networks,
 - net radio, and
 - message-switched networks.

3.2 Model Implementation and Experimentation

The next task in this project was to implement and execute a collection of PAWS/GPSM communications models for the purpose of evaluating the effectiveness of PAWS/GPSM

and ES/PAWS for communications networks. The models implemented and executed included the following:

- a) a data link control protocol
- b) a circuit-switched network with the following routing schemes:
 - static shortest path routing
 - a distributed adaptive algorithm
 - flooding
 - static shortest path routing in presence of failures
- c) a communications switching element.

The execution of these models demonstrated that it is feasible to model many aspects of communications systems using PAWS. Several issues were raised during this task, and some limitations were found in PAWS, many of which will be overcome in ES/PAWS. The modeling and experimental evaluation are discussed in detail in the Appendices.

3.3 Enhancement Specification

The last major task in this project was to specify the enhancements to PAWS/GPSM and ES/PAWS required for an effective tool for modeling Army communications networks. Many of the enhancements will become available as part of ES/PAWS later this year. The enhancement specifications were produced as follows:

- a) evaluating the shortcomings of the models described in section 5 and Appendix A
- b) studying the enhancements already underway as part of the ES/PAWS project, and
- c) developing specifications to bridge the gap between ES/PAWS and the Army modeling requirements.

The recommended enhancements are discussed further in section 6 and Appendix B.

3.4 Training

IRA will train CECOM staff in the use of the PAWS/GPSM communications models developed during this project. The purpose of this training is to allow the CECOM staff to gain familiarity with the use of such models, and to assist IRA in specifying modeling tool enhancements to be implemented in Phase II of this project.

4.0 Previous Related Work

There has been a substantial amount of modeling done in this area by IRA and IRA clients using PAWS and GPSM. We have drawn upon this experience during this study.

Some of this work indicates how PAWS and GPSM can be applied at any level of abstraction for modeling communication system components. At the physical layer of the OSI model [TAN 81], communication involves the transmission of bits over some (possibly error-prone) medium. A detailed model of the performance of an SNA communication line has been constructed using PAWS [DOR 84].

It is also possible to model an aspect of the OSI layer 1 and layer 2 interface, i.e., the communications hardware used to transmit and receive data from the communications medium. Typically this hardware is largely encapsulated in an IC chip that can also provide fairly sophisticated services such as zero insertion/deletion, generation of packet headers and check sums, and framing of incoming data. A model of a specialized high-speed VLSI communications chip has been constructed at IRA [JAI 87]. The chip consists of an 8-bit processor, high-speed serial communication controller, on-chip RAM and ROM, and interface to the host system CPU. The on-chip processor executes software for an OSI layer 2 (data link control) protocol. A model of the chip hardware was constructed in less than ten days and was verified in a few days by comparing emulation results with experimental data.

The performance of various layers of the OSI communications model has been studied. The performance of packet-switched inter-processor communications has been modeled [UPC 84] for a reconfigurable database machine architecture with dynamic connection of processor to memory via a regular SW-banyan blocking network. A model for predicting the peak and average response times for a cluster of VAX 11/780s at the link of a network with a star topology has also been constructed using PAWS [IRA 86]. The impact of communications services on overall system performance has been studied for a real-time point-of-sale system using PAWS [AND 84]. This model predicted significant performance improvement by increasing communication line speed from 4800 to 9600 baud.

Several studies by IRA and IRA clients have included modeling of failures in distributed systems. Performance models of a distributed real-time command and control system are

described in [FER 84, PAL 85]. These models include simulation of fault injection, propagation, and recovery. The models address the performance of transaction commit protocols in a distributed environment, and include submodels of a network interface, a long distance communication link, and a local area network (Ethernet). The model showed that resource contention at a remote host could cause as serious a performance degradation as a failure. It was also shown that an adaptive timeout mechanism can improve performance by minimizing "spurious" timeouts due to resource contention.

The performance of a token-passing ring and a reconfigurable lookahead network in the presence of failures has been modeled using PAWS [VEL 86]. This study also examines the performance of a failsafe distributed routing protocol in the presence of node and link failures. A study conducted by an IRA client models the performance of a Sperry 1100/44 based system with multiple command, arithmetic and peripheral processors used for real-time computations [CON 86]. This model considers the effect of failures on the execution of real-time tasks.

The experience gained from these previous studies has been very helpful in evaluating the feasibility of using C/PAWS for modeling Army communication systems. The work on failure modeling [FER 84, PAL 85, VEL 86, CON 86] has been particularly relevant for appreciating the effectiveness of PAWS in this area. These studies demonstrate that PAWS/GPSM can and has been used effectively for modeling many aspects of communications systems at any desired level of abstraction.

5.0 Modeling and Experimental Evaluation Results

The emphasis of the modeling and experimental evaluation task of this project was to determine the limitations of using the current versions of GPSM (2.3) and PAWS (3.0) for modeling communications systems. It should be noted that the objective of the experiments was not to construct production-quality simulation models. The effort was focussed on discovering the enhancements required to PAWS and GPSM, rather than on constructing sophisticated or efficient models.

Simulation models for the following communications system components have been constructed and evaluated for this study:

- 1) A data link control protocol
- 2) A circuit-switched communications network with the following routing schemes:
 - a) Static Shortest Path Routing
 - b) A Distributed Adaptive Algorithm
 - c) Flooding
 - d) Static Shortest Path Routing in presence of failures
- 3) A communications switching element

Although these models were constructed with certain types of communication networks in mind, they address issues common to a wide range of network modeling situations. The modeling and experimental evaluation report attached to this report discusses the models that were constructed, the simulation results obtained, and the insights gained from this experience. In this section we summarize the findings of the modeling task.

5.1 Data Link Control Protocol

The top-level GPSM graph for this model is shown in Fig. 1.

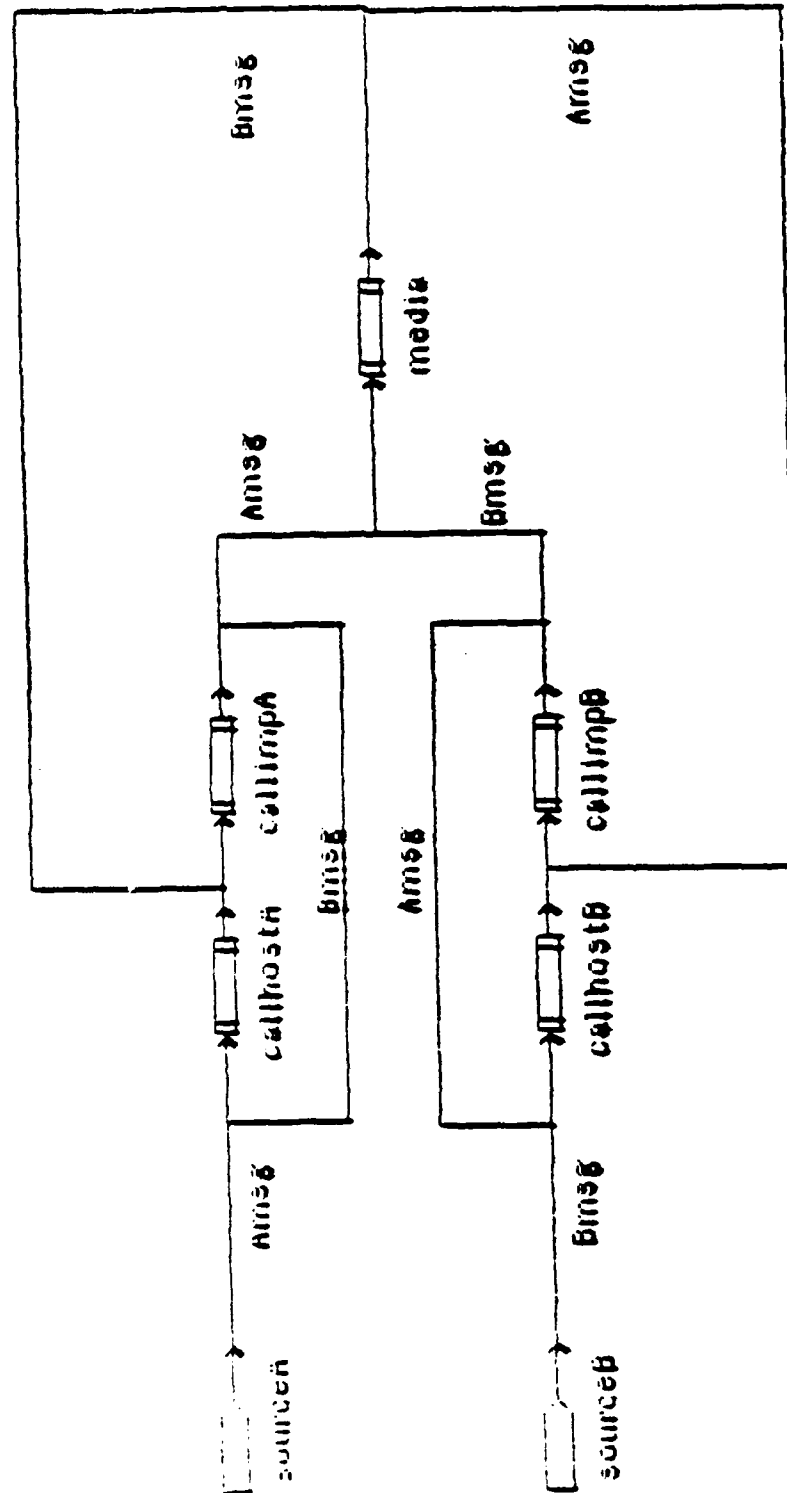


Fig. 1 Data link control protocol model: top-level graph.

It seems clear that the key aspects of data link protocols can be modeled in PAWS. This modeling will become even easier and more convenient in ES/PAWS, which will retain the declarative and visual programming aspects of PAWS and GPSM, and in addition provide the convenience and expressiveness of the C programming language.

An important aspect of this model is that it addresses issues such as timeouts, flow control, and message loss, which occur not only in the data link communications layer but also in other layers of the OSI model.

The model can be extended quite easily to deal with protocols with larger window sizes. If a specialized node type is developed to represent timeouts, as suggested in the enhancement specification, this type of modeling will become quite easy in C/PAWS.

5.2 Switching Network Model

The primary objective of the model is to investigate the effect of routing algorithms on the performance of circuit-switched networks. However, it should be noted that the performance issues involved in non-hierarchical routing in circuit-switched networks are very similar to those involved in routing for packet-switched networks [SCH 87]. Thus by restricting attention to non-hierarchical routing algorithms, the model can be used for both circuit-switched and packet-switched networks, possibly with a small amount of modification. The top-level GPSM graph for this model is shown in Fig. 2.

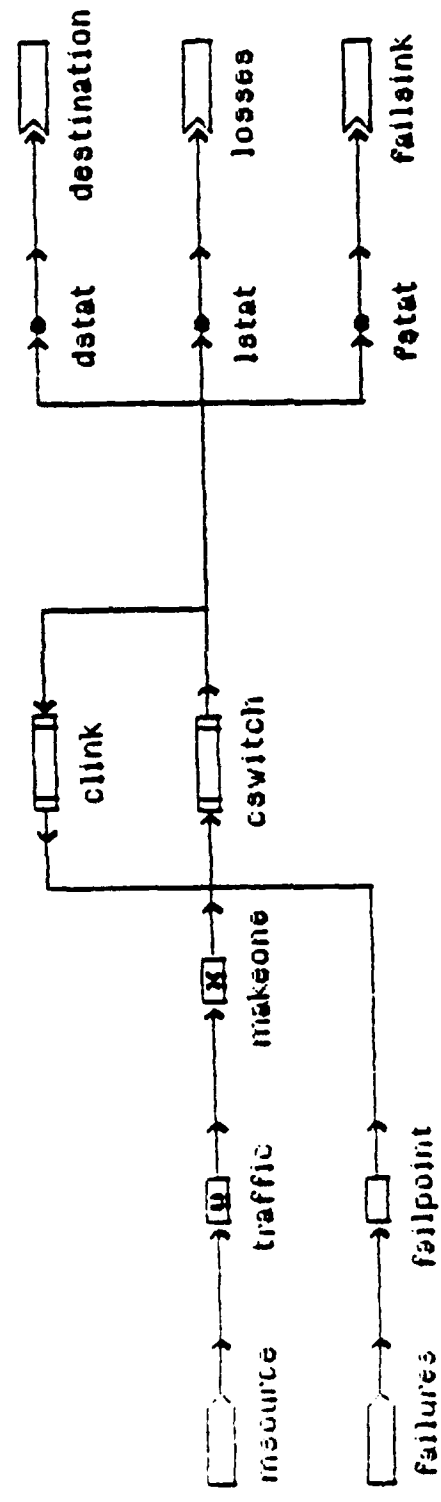


Fig. 2 Switched Network Model: top-level graph.

The network routing function is encapsulated in the PAWS submodel cswitch. Three different classes of routing algorithms were modeled. These represent very different types of routing criteria--ranging from minimizing response time to achieving maximum reliability. We have chosen to implement an instance of each type of algorithm in order to determine the feasibility of modeling that class. The algorithms modeled are:

- 1) Static Shortest Path Routing
- 2) A Distributed Adaptive Algorithm (Hot Potato)
- 3) Flooding
- 4) Static Shortest Path Routing with Failures

The routing algorithms were all run on the same example network [SCH 87].

It was found that all of these algorithms could be modeled fairly easily. All changes to the model were encapsulated in the submodel switch. The basic structure of this submodel remained the same for the different algorithms (except for the failure case), requiring changes only to the routing calculation.

Some of the issues raised by these models are summarized below:

- 1) C function utilities for calculating model parameters (eg. shortest paths) will be useful to the model developer
- 2) Hot potato routing was easy to implement as the key parameters required were conveniently available; in ES/PAWS this capability will be expanded.
- 3) Collecting response time statistics for an algorithm such as flooding can be made easier if more flexible FORK/JOIN constructs are provided in C/PAWS.
- 4) Verifying model correctness is not trivial, and support from the simulation tool is desirable.
- 5) It is important to distinguish between the model user and the model developer. The developer's interface should emphasize flexibility and power. The user's interface should emphasize simplicity.

- 6) Failure modeling can be modeled adequately.

5.3 Communications Switch Model

The switching element model studies the behavior of a switch in a circuit-switched network. The switch receives call requests from stations connected to it, and attempts to assign outgoing lines to satisfy these requests. Eventually all outgoing lines are allocated, and further incoming requests are called "lost calls". This model estimates connect time for a simple queued mode (lost calls delayed) circuit switch. The GPSM graph for this model is shown in Fig. 3.

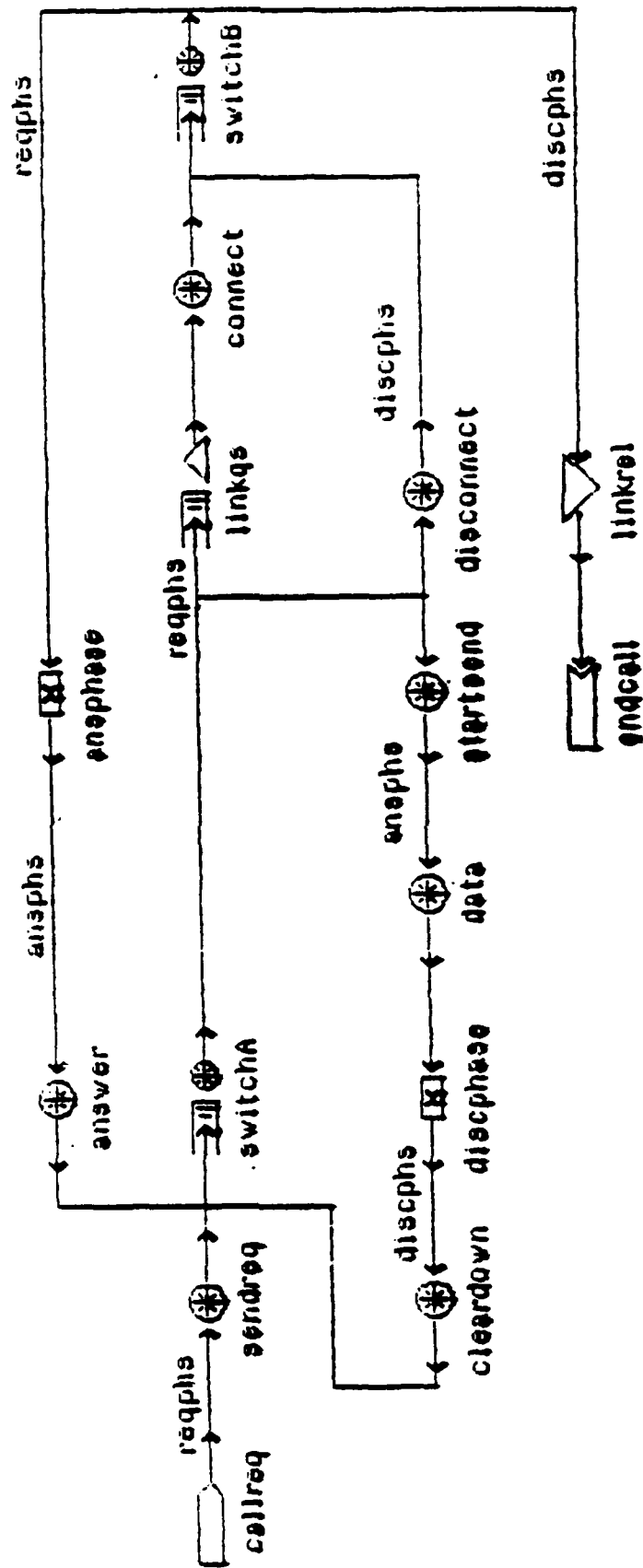


Fig. 3 Communications Switch Model

This model is a simple but fairly useful simulation of queued mode call processing for communication between two network switches. It is quite easy to generalize to the case where there are M switches with a different number of links between switch A and other switches by using arrays of SERVICE and ALLOCATE nodes. Since this can clearly be done within PAWS it was not pursued.

An interesting point is that to be accurate in the case where there is more than one switch to which switch A is connected, the "background" traffic in the network needs to be simulated. Since the call processing scenario is deterministic, the background processing can be calculated quite accurately for a certain network node, and fed as transactions that interfere with A-B communication at nodes switchA and switchB. This approach will give a more realistic estimate of network behavior, without the complexity of detailed simulation. In this way a much more complicated situation can be modeled with fairly small changes to the model.

6.0 Summary of Enhancement Specification

ES/PAWS and C/PAWS will be more flexible and powerful than the currently existing PAWS software and, in particular, will incorporate the expressiveness of the C programming language. Through the use of the highly user-friendly graphical interface, GPSM, and the use of libraries of submodels and procedures, a user will be able to simulate complex communications systems without needing, in general, to write code. Creating and maintaining the submodels and associated procedures, however, will require some familiarity with ES/PAWS syntax; these steps would be carried out by a model developer rather than a model user (see Appendix A, sec. 3.1).

Using C/PAWS the user will be able to obtain detailed simulation results; these results can then be input to other software packages such as SAS or SPSS to display the results of a single simulation run graphically (such as histograms of response times), as well as the results of several simulation runs (such as variations in response time as the offered load is increased upon each new run).

Some features of ES/PAWS particularly useful for communications modeling are:

- access to the C language (data structures, expressions, functions, etc.)
- macro pre-processor
- submodel parameters
- finite-state machines
- enhanced version of GPSM, including:
 - structured input facility (SIF)
 - graph library facility
 - scrolling GPSM graphs
 - improved on-line help
 - critique of GPSM graphs

The proposed specific C/PAWS enhancements include:

- 1) Forms and menus for the model user interface
- 2) User-defined icons for specialized components
- 3) Convenient specification of statistics collection, including statistics involving submodels
- 4) Processing of simulation output for graphical display
- 5) A TIMEOUT node type for modeling communications protocol features
- 6) An Interrupt Resume node type for making failure modeling easier. This fits into an overall failure modeling methodology
- 7) Libraries of C functions for analytic bottleneck analysis and routing algorithm calculations
- 8) Libraries of re-usable submodels encapsulating common communications subsystems
- 9) Specification of integrity constraints and reasonableness checks for nodes, transaction states, submodels and simulation results
- 10) More flexible FORK and JOIN constructs
- 11) Explicit control of the generation of transactions at SOURCE nodes, allowing a user to control a simulation run more closely
- 12) Support for hierarchical simulation

7.0 Conclusions

This project has convinced us that it is indeed feasible and practical to satisfy Army communications modeling requirements by enhancing ES/PAWS according to sections 6.0 and Appendix B. We believe that such an enhancement project will produce a commercial product that will dramatically improve the design and analysis of military communications networks as well as non-military networks. Some of the enhancements are already underway as part of the ES/PAWS project. The other enhancements await Phase II of this project.

8.0 References

AND 84

G. E. Anderson "The Coordinated Use of Five Performance Evaluation Methodologies", Communications of the ACM, vol. 27, no. 2, 1984.

BRO 88

J. C. Browne, P. Jain, D. M. Neuse and M. Esslinger, "ES/PAWS - A System Level Design Aid", submitted for publication in *Proceedings of the Design Automation Conference*, June, 1988.

CON 86

Customer confidential document. The model involves a Sperry 1100/44 computer system with multiple command, arithmetic and peripheral processors used for real-time computations. Permission is being sought to disclose this information.

DOR 84

Vladimir Dorfman, "SNA Communication Line Performance Analysis", Fujitsu Systems of America, Technical Report 92-00024, July, 1984 (proprietary).

FER 84

V. Fernandes, J. C. Browne, D. Neuse, and R. Velpuri, "Some Performance Models of Distributed Systems", *Proceedings of the CMG XV International Conference*, Dec., 1984.

HAM 86

J. L. Hammond and P. J. P. O'Reilly, *Performance Analysis of Local Computer Networks*, Addison-Wesley, 1986.

IRA 86

"PAWS Performance Models of a Computer Network Hub," Information Research Associates, Internal Document, 1986.

IRA 87a

Information Research Associates, *Performance Analyst's Workbench System (PAWS) User's Manual*, 1987.

IRA 87b

Information Research Associates, *Graphical Programming of Simulation Models (GPSM) User's Manual*, 1987.

IRA 88

Information Research Associates, "Requirements Specifications for C/PAWS", Task 1 report of U. S. Army CECOM Phase 1 SBIR project, submitted to CECOM on January 6, 1988.

IAF 82

M. Jaffe and F. H. Moss, "A Responsive Distributed Routing Algorithm for Computer Networks", *IEEE Trans. On Comm.*, Vol. COM-30 no. 7, pp. 1758-1762, July 1982.

JAI 87

Prem Jain, "Architecture Design of a VLSI Chip Using PAWS/GPSM", Technical Report, Information Research Associates, July 1987.

LAW 82

A. M. Law and W. D. Kelton, Simulation Modeling and Analysis, McGraw-Hill, 1982

NAV 87

United States Navy Contract No. N60021-86-C-0145, High Level Simulation of Electronic Systems, 1987.

PAL 85

Annette Palmer, J. C. Browne, J. Silverman, A. Tripathi, and R. Velpuri, "A Performance Model of a Fault-Tolerant Distributed System for Evaluating Reliability Mechanisms", *Proceedings of the CMG XVI International Conference*, 1985.

SCH 87

Mischa Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*, Addison-Wesley, May 1987.

TAJ 77

W. D. Tajibnapis, "A Correctness Proof of a Topology Information Maintenance Protocol for Distributed Computer Networks", *Comm. ACM*, vol. 20, pp. 477-485, July 1977.

TAN 81

Andrew Tanenbaum, *Computer Networks*, Prentice-Hall, 1981.

UPC 84

E. Upchurch, "Modeling Packet Switched Interprocessor Communications", Proceedings of the 15th Annual Modeling and Simulation Conference, University of Pittsburgh, 1984.

VEL 86

Rajkumar Velpuri, "Performance Study of Zeus Distributed System with Different Communication Networks", *M. S. Thesis*, The University of Texas at Austin, May, 1986.

Appendix A

Modeling and Experimental Evaluation Report

Contents

1.0	Introduction	A1
2.0	Simulation Models.....	A3
3.0	Data Link Control Protocol Model.....	A4
3.1	The Protocol.....	A4
3.2	Modeling Approach.....	A4
3.3	Protocol Model	A5
3.4	Flow Control.....	A7
3.5	Timeout.....	A9
3.6	Simulation Results.....	A11
3.7	Summary.....	A13
4.0	Switching Network Model	A14
4.1	Network Model	A14
4.2	Routing Algorithms	A17
4.3	Static Directory Routing.....	A19
4.3.1	Shortest Path Trees.....	A19
4.3.2	Routing model	A21
4.3.3	Simulation Experiments	A24
4.3.4	Variations on Shortest Path Routing.....	A26
4.3.5	Discussion.....	A26
4.4	Distributed Adaptive Routing.....	A27
4.4.1	Hot Potato Routing Model	A28
4.4.2	Simulation Results.....	A31
4.4.3	Discussion.....	A33
4.5	Flooding.....	A34
4.5.1	Routing model	A34
4.5.2	Discarding Duplicates.....	A38
4.5.3	Network Overflow.....	A40
4.5.4	Simulation Results.....	A41
4.5.5	Discussion.....	A41

Contents (Cont'd)

4.6	Shortest Path Routing With Failures	A42
4.6.1	Failure Model.....	A42
4.6.2	Simulation Experiments	A44
4.6.3	Adaptive Shortest Path Routing.....	A46
4.6.4	Discussion.....	A50
5.0	Communications Switch Model	A51
5.1	Switch Model.....	A53
5.2	Simulation Experiments	A55
5.3	Discussion.....	A57
6.0	Conclusions.....	A58
References	A59

List of Figures

Fig. 1A	Data Link Control Protocol Model: Top-Level Graph
Fig. 1B	Data Link Control Protocol: Flow Control Modeling
Fig. 1C	Data Link Control Protocol: Timeout Modeling
Fig. 1D	Response Time Statistics for Data Link Control Protocol
Fig. 2A	Switched Network Model: Top-Level Graph
Fig. 2B	Example Network
Fig. 3A	Shortest Path Tree
Fig. 3B	Static Shortest Path Routing
Fig. 3C	Static Shortest Routing Data Structures
Fig. 3D	Response Time Statistics for Shortest Path Routing
Fig. 4A	Hot Potato Routing: Switch Model
Fig. 4B	Significant Data Structures for Hot Potato Routing
Fig. 4C	Response Time Statistics for Hot Potato Routing Algorithm
Fig. 5A	Flooding: Modified Network Model
Fig. 5B	Flooding: Switch Model
Fig. 6A	Failure Modeling for Shortest Path Routing Switch Model
Fig. 6B	Response Time Statistics for Shortest Path Routing with Failure
Fig. 6C	Switch Model for Adaptive Routing in the Presence of Failures
Fig. 7A	Call Processing Scenario
Fig. 7B	Communications Switch Model
Fig. 7C	Connect Time Statistics for Queued Mode Circuit Switch

CECOM COMMUNICATIONS MODELING PROJECT PHASE I MODELING AND EXPERIMENTAL EVALUATION REPORT

1.0 Introduction

The objective of the modeling and experimental evaluation task of this Phase I project is to determine the feasibility of using IRA's PAWS and GPSM simulation tools [IRA 87a, IRA 87b] for modeling the performance of Army communication systems. The approach taken is to construct PAWS/GPSM models for a variety of representative communication systems used by the Army, and evaluate the limitations encountered in the simulation tools while doing so. The experience gained from these experiments will then be used to define the enhancements required to the simulation tools in order to model Army communications system easily and efficiently. These enhancements will be carried out in Phase II of this project, with the resulting advanced simulation tool for modeling communications systems called C/PAWS. The overall approach to developing C/PAWS is to adapt ES/PAWS, a top-down design tool for electronic systems being developed for the Navy [NAV 87, BRO 88]. ES/PAWS will be adapted to modeling communication systems, using the knowledge and experience gained during this Phase I project.

The emphasis of the modeling and experimental evaluation task of this project is to determine the limitations of using the current versions of GPSM (2.3) and PAWS (3.0) for modeling communications systems. It should be noted that the objective of the experiments discussed in this report is not to construct production-quality simulation models. The effort is focussed on discovering the enhancements required to PAWS and GPSM, rather than on constructing sophisticated or efficient models.

Several simulation models have been constructed which reflect many issues in communication systems performance modeling in general, and Army networks in particular. The models study features common to several network layers (e.g., timeouts and flow control) as well as several switching methods (e.g., routing for circuit, packet and message-switched networks). An important aspect of the experience that IRA has gained in this area is expertise in modeling the effect of failures on communications systems performance. This work has previously been published [FER 84, VEL 86] and drawn upon in this study due to its special relevance to the Army.

The following sections describe each of the models and routing algorithms developed in this study and the results of using these models to perform some simple simulation experiments. This is followed by a brief discussion of the experience gained during the experimentation. Extensive simulations were not carried out as it was felt that they would not provide any new insights into the enhancements required for C/PAWS. A detailed discussion of the limitations encountered in PAWS and the enhancements required will be presented in the enhancement specification (Task 2) report of this project.

2.0 Simulation Models

Simulation models for the following communications system components have been constructed and evaluated for this study:

- 1) A data link control protocol
- 2) A circuit-switched communications network with the following routing schemes:
 - a) Static Shortest Path Routing
 - b) A Distributed Adaptive Algorithm
 - c) Flooding
 - d) Static Shortest Path Routing in presence of failures
- 3) A communications switching element

As explained later in this report, although these models were constructed with certain types of communication networks in mind, they address issues common to a wide range of network modeling situations.

In addition to the models listed above, PAWS and GPSM have been used by IRA and IRA clients in order to model the following:

- 1) The CSMA/CD protocol (Ethernet) [FER 84]
- 2) A token-passing ring [VEL 86]
- 3) A hardware reconfigurable look ahead network [VEL 86]
- 4) An SNA communication line [DOR 84]
- 5) A VLSI Communication chip [JAI 87]
- 6) Packet-switched inter-processor communication [UPC 84]
- 7) A VAX 11/780 cluster at the hub of a star network [IRA 86]
- 8) The impact of communication services on a real-time point-of-sale system [AND 84]
- 9) A failsafe distributed routing protocol in presence of failures [VEL 86]
- 10) Failures in a Sperry 1100/44 - based system used for real-time calculations

Since these models have been discussed elsewhere they will not be discussed in this report.

3.0 Data Link Control Protocol Model

The one-bit sliding window data link protocol was an interesting example that yielded some useful insights. These protocols are important in the analysis of communications systems performance.

Some limitations were found in the ease with which PAWS can be used to model these types of data link control protocols. However, it was noted that these limitations can be overcome in a fairly straight-forward manner, and in fact the ES/PAWS simulation tool under development will overcome most of these.

The protocol is briefly described below, followed by the modeling approach, the model and some simple simulation results.

3.1 The Protocol

The protocol was chosen from the data link control chapter in Tanenbaum's text ("Computer Networks"), where it is referred to as "protocol 4" [TAN 81]. This protocol uses a one-bit sliding window to provide full-duplex data transmission in the presence of message corruption, loss, and duplication. Piggybacked positive ACKs are used for message acknowledgement. The protocol assumes two stations communicating over a point-to-point link. Each station consists of a host (message source) and an IMP (network interface). Each IMP maintains global variables which contain the windowing information. Every message sent over the link contains a (1-bit) sequence number and the next sequence number expected from the other side. The latter is essentially an ACK.

3.2 Modeling Approach

The protocol was modeled in a top-down hierarchical fashion to try to match a communications engineer's view of the problem rather than a performance analyst's approach. Thus the top-level GPSM graph is highly intuitive, consisting entirely of source nodes and submodel calls. Although this is a somewhat inefficient way of modeling the protocol, it is modular and preserves information hiding.

A limitation of PAWS is its restrictive submodeling facility. The submodeling facility incorporated in ES/PAWS will have a clean and convenient interface, similar to the

semantics of procedure calls in conventional programming languages. It was discovered that it is not convenient to define a re-usable library of submodels in PAWS 3.0. For this reason a collection of models rather than submodels will be delivered to CECOM, as we believe they will be more useful to CECOM at the present time.

The protocol allows IMPs to accept messages from the hosts only when they have processed an ACK for a previous message. In Tanenbaum's description the flow control required to prevent a host from swamping an IMP with messages is assumed to take place in the host, and is not described. For the purposes of modeling, host-IMP flow control was explicitly specified.

3.3 Protocol Model

The protocol model is described with the aid of the top-level GPSM graph (Fig. 1A). The model is symmetrical in that the functionality of the two communicating entities, A and B, is identical; they differ only in the data structures they operate upon. (In ES/PAWS it will be easier to express this similarity using submodel calls, in a fashion similar to calling library procedures in a conventional programming language). In the following description the operation of side A is described -- side B is similar.

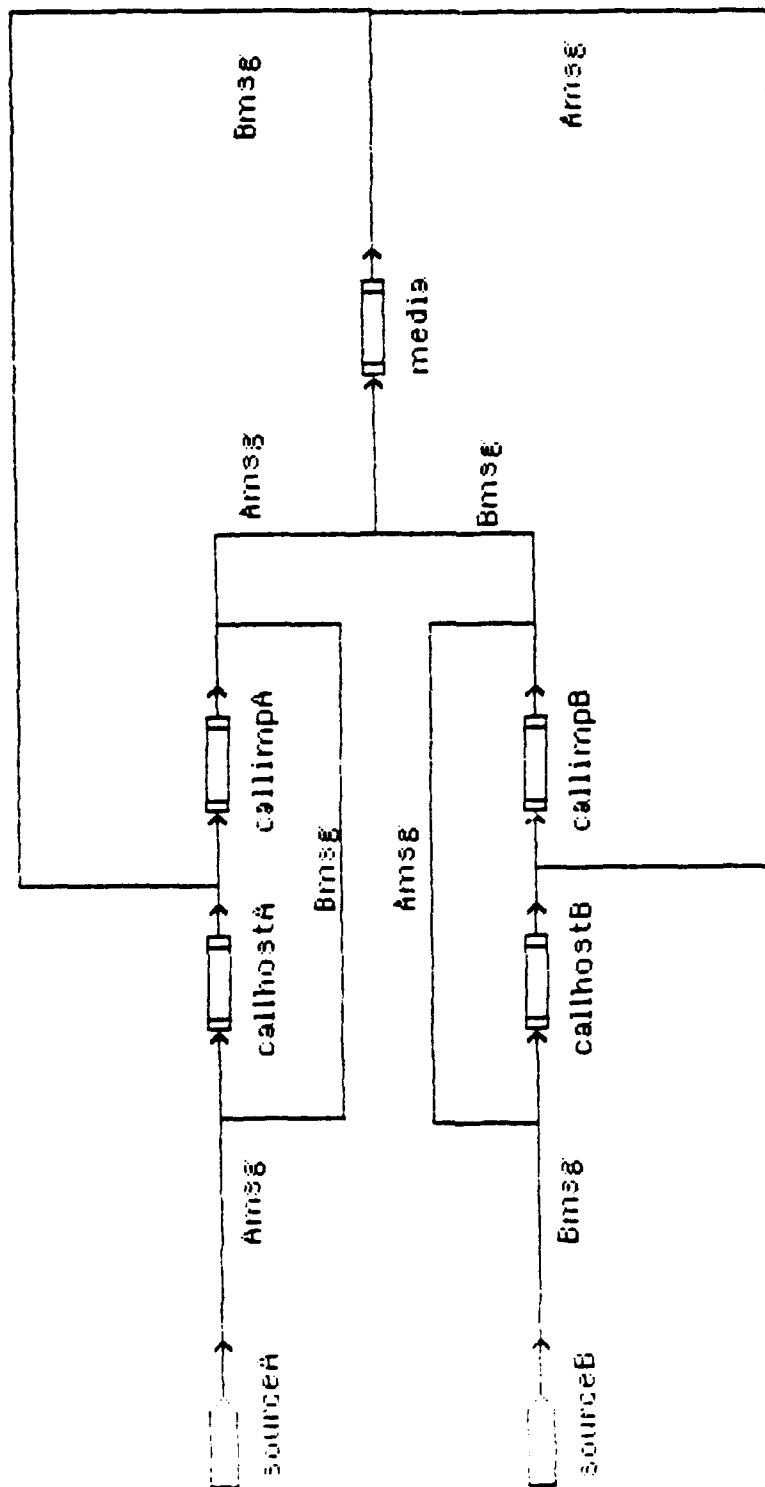


Fig. 1A Data link control protocol model: top-level graph.

Node sourceA generates messages from the network host A. These are processed in the host (at callhostA) before being forwarded to the network interface (callimpA). The submodel called from callhostA essentially models the host-IMP flow control, and its functionality is described in sec. 3.4.

The IMP handles both outgoing messages from the host as well as incoming messages from the media. For outgoing messages it performs the windowing function by setting the 1-bit sequence and acknowledgement numbers. It also performs the timeout and retransmission function, described in sec. 3.5. For incoming messages, the IMP checks to see if they carry the correct sequence number, ACK, or both. If an ACK is received, it prevents a timeout as described in sec. 3.5. The received message is then forwarded to hostA to open the host-IMP flow control.

The key modeling issues are flow control and timeouts, which are discussed below.

3.4 Flow Control

A natural way to implement flow control is via PAWS tokens. Modeling host-IMP flow control required four ALLOCATE and RELEASE nodes, arranged as in Fig. 1B. Note that this arrangement does not require transactions to carry the token through the entire model. Thus the token management function can be encapsulated within this one submodel.

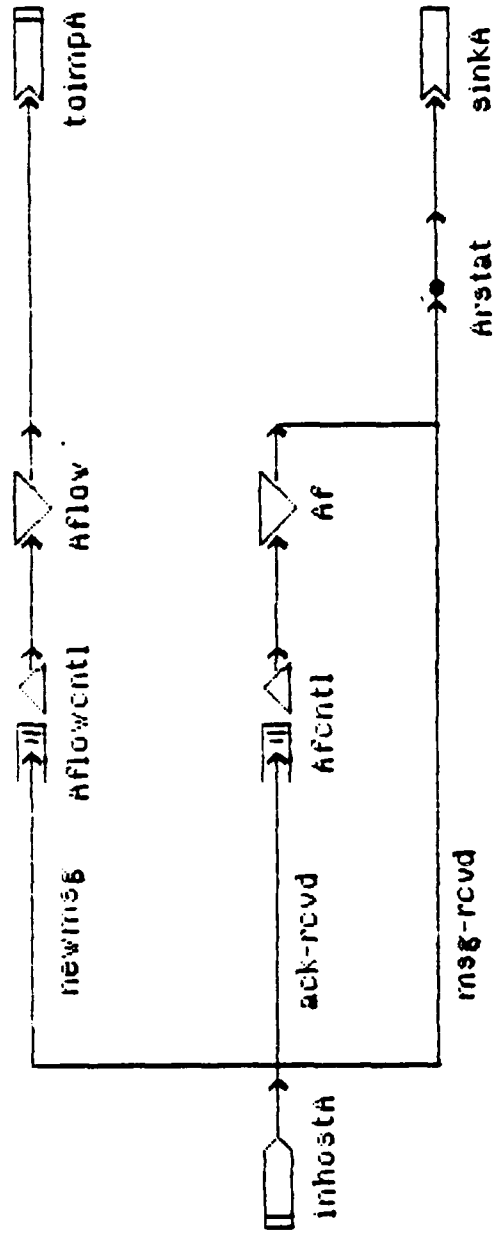


Fig. 1B Data link control protocol: Flow control modeling

Node Aflowcntl initially holds one token. An incoming message obtains the token and, upon reaching node Aflow, releases it to node Afcntl, thus preventing any new messages from the host onto the link. An ACK from the other side uses the token at Afcntl and releases it at node Af to Aflowcntl, thus allowing any queued messages there onto the link. The number of tokens placed initially at node Aflowcntl is equal to the protocol window size.

Flow control in communication systems is intended to prevent a fast sender of messages from overflowing the capacity of a slow receiver. Two schemes are common. In the first, the sender produces messages up to a fixed predetermined number and waits for permission from the receiver to send any more. This is the scheme used in this example. In the second, the sender continues to send messages until instructed by the receiver (via a choke packet, XOFF signal, or CTRL-S character) to stop doing so; the sender resumes upon receiving an explicit signal from the receiver.

In C/PAWS and ES/PAWS it is possible to model both these schemes using two nodes. The first can be modeled using an ALLOCATE and CREATE node pair, while the second can be modeled using a SERVICE node and a SET node. Both these mechanisms are quite general and convenient, so it is not felt necessary to introduce a new node type for modeling flow control.

3.5 Timeout

A natural way to specify timeout processing is via delay nodes and PAWS interrupts. In Fig. 1C an example of how this can be done is shown (in the actual protocol model this function is contained in two separate graphs — the non-essential details are not shown in Fig. 1C).

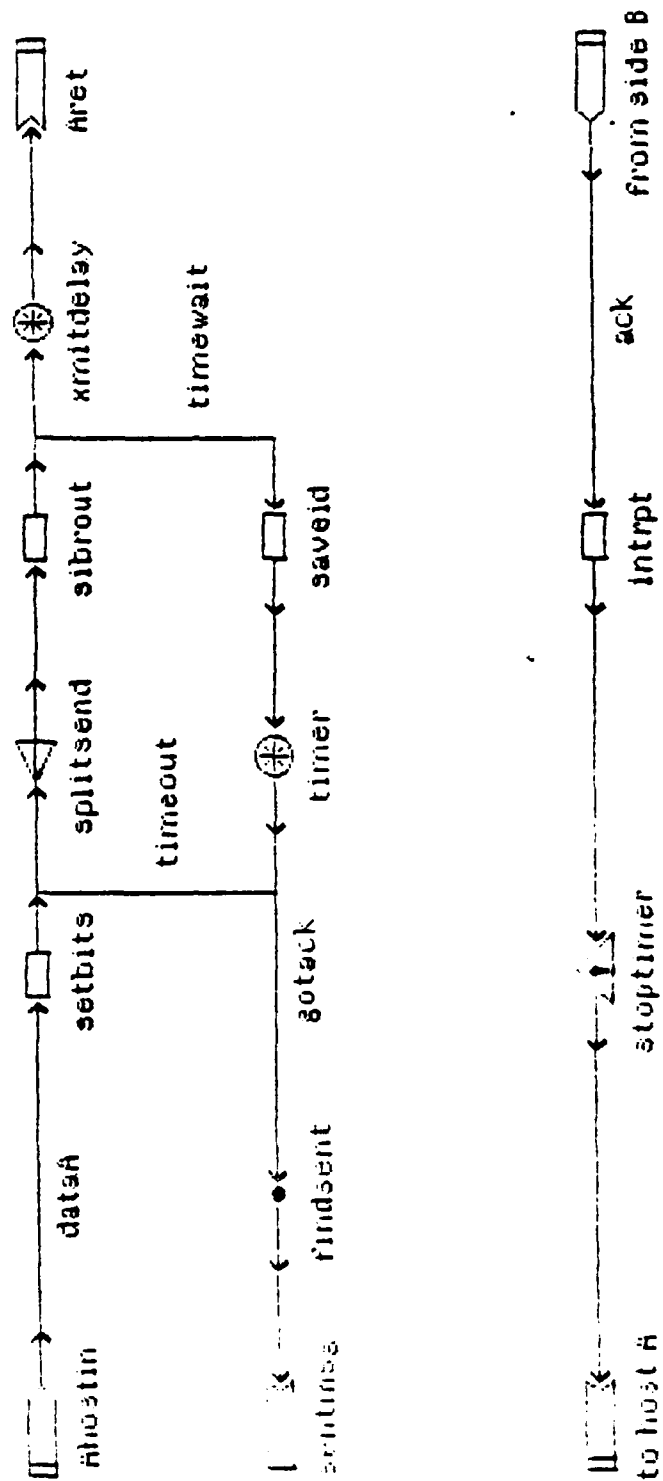


Fig. 1C Data link control protocol: Timeout modeling.

A message arriving at node splitsend is copied so as to create two identical siblings. At node sibout, one sibling is routed out to the link, while another is routed to a timer to await an ACK. At node saveid the ID of the waiting transaction is recorded. If the transmitted sibling correctly reaches the other side an ACK is sent, which reaches node intrpt. Here the information saved by node saveid is used to determine which transaction is awaiting the ACK. At node stopimer the ACK interrupts the timer and routes that transaction to a sink.

If an ACK is not received within the specified Timeout, the waiting transaction leaves node timer and is again split into two siblings at node splitsend, in order to cause a retransmission. This will continue until an ACK is received.

If timeouts are to be specified frequently, a timeout node type, whose semantics have been carefully defined to include retransmissions of multiple outstanding messages, may be defined.

3.6 Simulation Results

Each of the two hosts in the model generates messages with an exponential inter-arrival time distribution of 20 time units. The cable propagation delay is a constant 5 units. There is 1.5 time unit constant processing delay per message, composed of two delays of 0.5 units each at the source destination IMP's (nodes crunch and hostxmit) and 0.5 units for processing transmission of each message copy (node xmitdelay). There is an 85% probability that a message traverses the cable without error. Message corruption and loss account for the rest. Each IMP has a Timeout variable which determines how long it waits for an ACK before retransmitting the last message. The Timeout variable is the same for both nodes. The model is simulated for 5000 time units. Although this is a fairly short simulation run (only about 250 messages are generated per side) it is sufficient to get an estimate of protocol behavior.

The model was run for 5000 time units with a value of 25 units for the Timeout variable. The response time from node Ahostin to statistics collection node findsent (Fig. 1C) is shown as a histogram in Fig. 1D. This appears reasonable given the deterministic processing delays.

RESPONSES FROM AHOSTMSG /AHOSTIN (1) TO AHOSTMSG /FINDSENT (1)

CATEGORY: /MSG

RESPONSE-TIME INTERVAL	NUMBER IN INTERVAL	Z IN INTERVAL	HISTOGRAM
0.000 <= X < 5.000	0.000	0.00	I<
5.000 <= X < 10.000	0.000	0.00	I<
10.000 <= X < 15.000	156.000	70.91	I*****I<
15.000 <= X < 20.000	2.000	0.91	I<
20.000 <= X < *INFINITY*	62.000	28.18	I*****I<
TOTAL: 220.000			

SUMMARY

MEAN: 21.316 2ND MOMENT: 683.670
VAR: 229.302 STNDRD DEV: 15.143

Fig. 1D Response Time Statistics for Data Link Control Protocol

3.7 Summary

It seems clear that the key aspects of data link protocols can be modeled in PAWS. This modeling will become even easier and more convenient in ES/PAWS, which will retain the declarative and visual programming aspects of PAWS and GPSM, and in addition provide the convenience and expressiveness of the C programming language.

An important aspect of this model is that it addresses issues such as timeouts, flow control, and message loss, which occur not only in the data link communications layer but also other layers of the OSI model.

The model can be extended quite easily to deal with protocols with larger window sizes. If specialized node types are developed to represent timeouts and flow control, this type of modeling will become very easy in C/PAWS.

4.0 Switching Network Model

This section describes a model of a switched communications network. The following sections describe the modeling of various routing algorithms for this network.

The primary objective of the model is to investigate the effect of routing algorithms on the performance of circuit-switched networks. However, it should be noted that the performance issues involved in non-hierarchical routing in circuit-switched networks are very similar to those involved in routing for packet-switched networks [SCH 87]. Thus by restricting attention to non-hierarchical routing algorithms, the model can be used for both circuit-switched and packet-switched networks, possibly with a small amount of modification.

Since the emphasis is on the feasibility of modeling routing, other network issues, e.g. congestion control, have been ignored. It seems clear that if routing can be modeled using PAWS and GPSM, other network layer issues can also be handled. In addition, a data link protocol model (sec. 3) indicates that features common to several communications layers, e.g. timeouts and flow control, can be modeled using PAWS and GPSM.

An important aspect of the network example is that it models the effect of failures on performance. The example also incorporates the effect of repairing failed components.

4.1 Network Model

The network model will be explained with the aid of the top-level GPSM graph (Fig. 2A). Note that for the purpose of performance analysis a communications network can be conceptualized basically as messages circulating between network switching nodes and communications links. In the model, these are represented by the submodels cswitch and clink respectively. This performance modeling diagram does not represent the network topology. It is an abstraction of a general network, where messages continuously travel from switches to links until they reach the destination host.

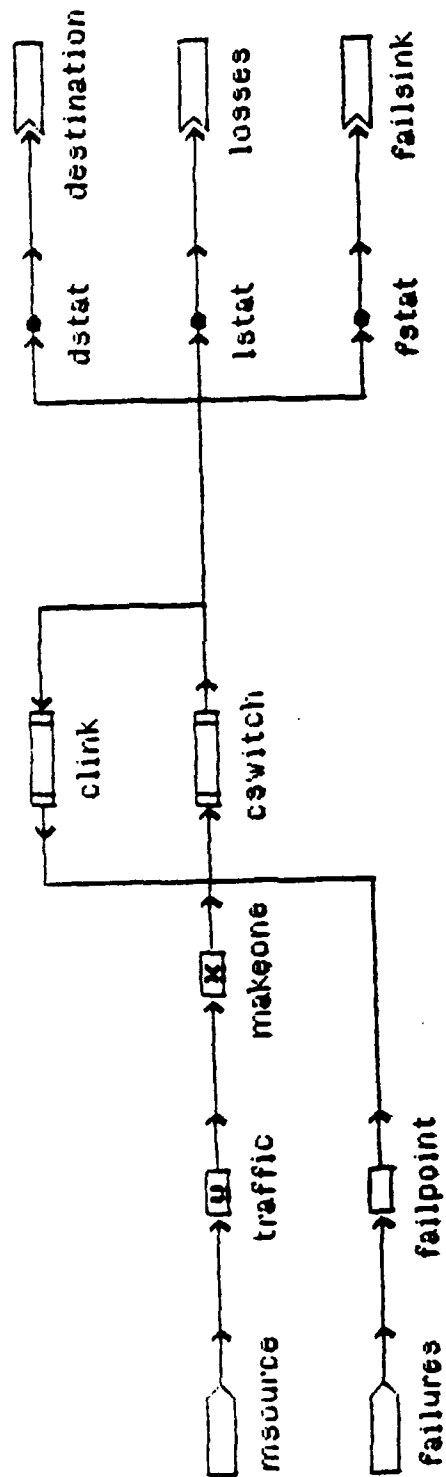


Fig. 2A Switched Network Model: 'top-level graph.'

The node msource models the generation of network messages for all hosts in the network. Messages are generated at a different rate for each host, as determined by a source rates data vector which is initialized in the PAWS "run" section. The messages generated at msource only contain the ID of the source host but not that of the destination host. At node traffic the network connectivity, link delay and traffic probability matrices are read in from a data file (this is only done once per simulation run). Messages are assigned destination host ID's based upon the traffic matrix. Thus a message reaching a network switch (modeled by PAWS node cswitch) carries its source and destination host IDs. At cswitch the network routing function is carried out. Messages leaving cswitch carry the ID of the next switch to which they are being forwarded and the link over which they will be sent. The amount of delay that the message will experience while traversing this link is also calculated here, although the actual delay is modeled at node clink.

Messages that reach their destination will leave the model via the sink node destination. Messages forwarded to other switches enter node clink where they will experience delay, and possibly, loss and corruption, before returning to cswitch. Lost and corrupted messages leave the model via sink node losses.

The nodes failures, failpoint and failsink are used for modeling failure. Failure processing will follow the approach taken in previous IRA studies of failure modeling for distributed systems [FER 84, VEL 86]. In sections 4.3 to 4.5 several routing algorithms will be described without reference to failures; the failure transactions essentially enter and leave the network without having any effect, and can be ignored. In sec. 4.6 a model will be described that considers the effect of failures; failure processing is discussed in detail there.

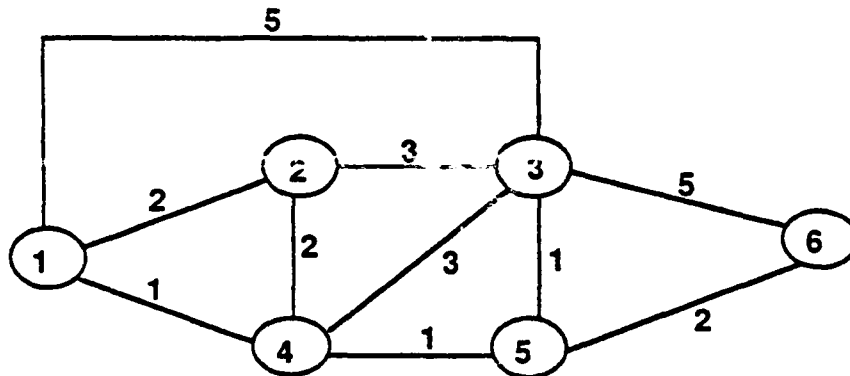
4.2 Routing Algorithms

The network routing function is encapsulated in the PAWS submodel cswitch. Three different classes of routing algorithms are described below. These represent very different types of routing criteria--ranging from minimizing response time to achieving maximum reliability. We have chosen to implement an instance of each type of algorithm in order to determine the feasibility of modeling that class. The algorithms modeled are:

- 1) Static Shortest Path Routing
- 2) A Distributed Adaptive Algorithm
- 3) Flooding
- 4) Static Shortest Path Routing with Failures

The routing algorithms were all run on the same example network [SCH 87]. The network topology and various parameter vectors are shown in Fig. 2B. In the traffic probability matrix, entry $T(i, j)$ represents the probability that a message generated at host i has host j as its destination. The source rate vector was decided upon somewhat arbitrarily. It does have the effect that with the traffic matrix shown in Fig. 2B, over one fifth of the total network traffic goes from (source) host 1 to (destination) host 2.

Example Network



Circles represent nodes in the network.

Lines represent bidirectional links, with link delays.

Source rate vector (arbitrary choice of rates):

Node	Inter-arrival time for message
1	10
2	20
3	30
4	40
5	50
6	60

Traffic Probability Matrix, T to node

	1	2	3	4	5	6
1	0.0	0.5	0.2	0.1	0.0	0.2
2	0.1	0.0	0.2	0.1	0.4	0.2
3	0.2	0.1	0.0	0.1	0.2	0.4
4	0.3	0.1	0.2	0.0	0.3	0.1
5	0.3	0.1	0.2	0.3	0.0	0.1
6	0.1	0.2	0.2	0.4	0.1	0.0

Fig. 2B

4.3 Static Directory Routing

This is a simple algorithm and one of the most widely used [TAN 81]. Each network switch maintains a table with one row for each possible destination host. A row gives the outgoing line (or next switch) that messages for that destination should be forwarded on. There may be several possible outgoing lines, ranked in order of preference. Thus routing depends upon the destination host ID, and outgoing lines can be chosen by some metric so as to optimize performance. Typical metrics include link delay and hop count.

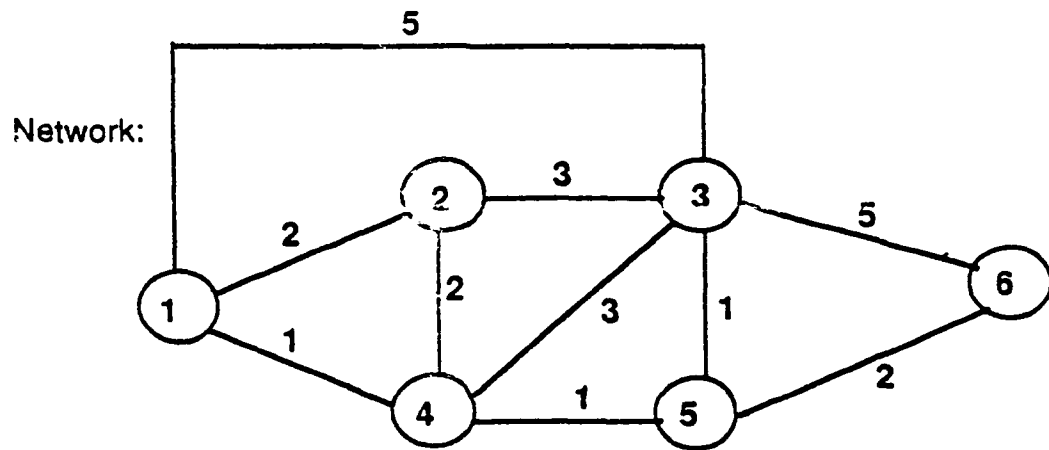
The main advantage of static directory routing is that it is simple to understand and implement. Typically network managers calculate the routing tables and load them into the switches manually. Static routing gives good performance if the network topology and traffic do not change much. The main disadvantage is that the algorithm does not adapt to changes in traffic.

4.3.1 Shortest Path Trees

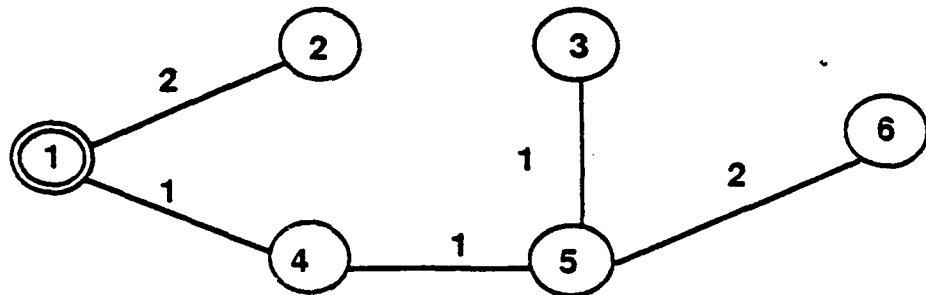
In our implementation of static routing each switch chooses the outgoing line that has minimum delay for each destination host, i.e., link delay is used as the path metric. Notice that the optimality principle applies : if the shortest path from switch I to switch K is via switch J then the shortest path from J to K also falls along the same route. Thus the set of optimal routes from a source to all destinations forms a tree rooted at the source. We will call this a shortest path tree. Since failures are not being modeled in this particular simulation each switch chooses only one outgoing line per destination.

The shortest path trees, with link delay as metric, are shown in Fig. 3A for nodes 1 and 2 of our example network.

Shortest Path Trees



Shortest Paths from Node 1 to other nodes



Shortest Path Tree, Node 2

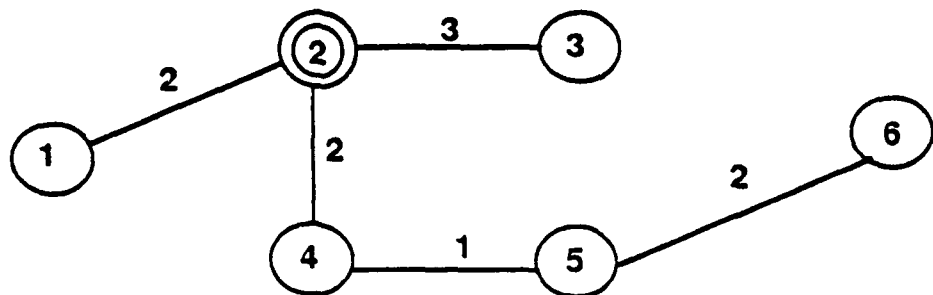


Fig. 3A

4.3.2 Routing model

The routing function is carried out in the submodel switch (see Fig. 3B). Data messages enter the token release node linkrel where they release access to the communication link over which they arrived. Since messages generated at the host (msource) have not yet entered the network, they do not have any links to release, and proceed to node route, which invokes a FORTRAN program to perform the routing calculation.

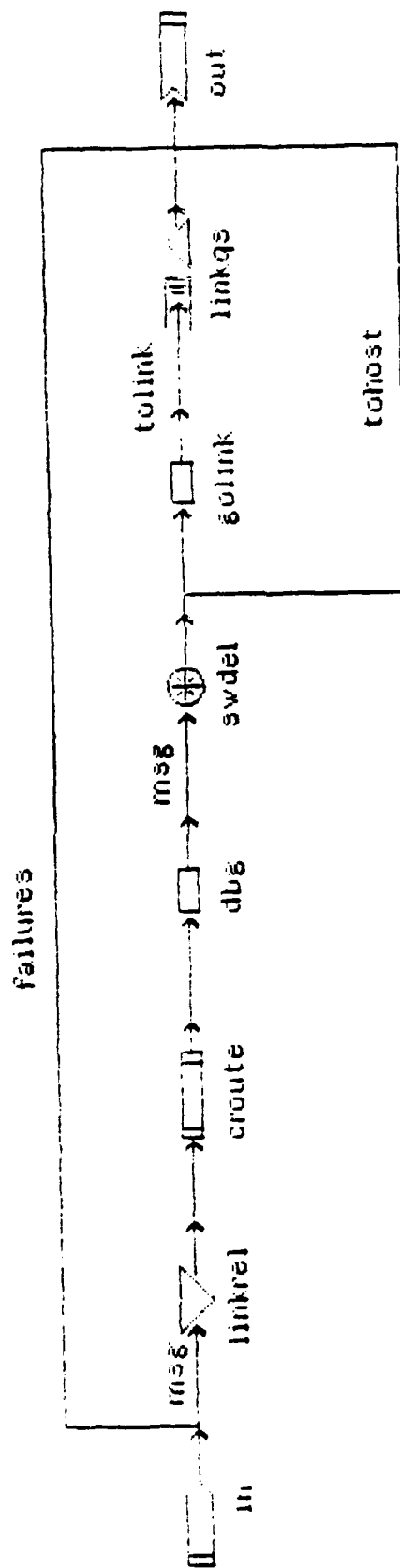


Fig. 3B Static Shortest Path Routing: Switch Model.

The routing program expects as input the switch to which the message has arrived, and the ultimate destination of the message. Depending on the destination the program searches a routing matrix to determine the next switch to which the message should be forwarded based upon the shortest path tree for this switch. When the message leaves croute it carries the IDs of the current switch and the next switch in its local variables, as well as the amount of delay it will experience in going to the next switch. (Only the amount of the delay is calculated here--the actual delay occurs in the submodel link). The routing program also determines whether the message has reached its ultimate destination, and if so, sets a local flag in the message.

The data message then experiences a small switching delay at swdel. This delay has an exponential distribution, to roughly model the contention for processing that occurs when several messages are present at the switch. Since the delay is usually much smaller than the link delays, it is not modeled in detail. If the message has reached its destination it leaves the switch via arc tohost. Otherwise, at node golink the message is assigned the actual communication link number over which it must travel in order to reach the next switch. Since only one message may travel on any link at any time, at node linkqs the message contends with other messages which need the same link. The linkqs icon represents an array of N^2 PAWS ALLOCATE nodes, where N is the number of network hosts. There is one token per ALLOCATE node. A message bound for link i receives a token at linkqs(i) and releases it, after it has travelled through clink, at node linkrel, allowing the next message queued at node linkqs (i) access to link i .

In order to aid understanding of the PAWS code, the variables used in this model are documented in Fig. 3C.

PAWS local integers

- LI [1] : the message source node ID
- LI [2] : the message destination ID
- LI [3] : current switch ID
- LI [4] : next switch ID
- LI [5] : delay to experience in going to next switch
- LI [6] : the communication link number
$$= (LI [3] - 1) * (\text{number of nodes}) + LI [4]$$

PAWS local booleans

- LB [1] : message is to be routed to a link
- LB [2] : message has reached destination
- LB [3] : message leaves model as a loss

Fig. 3C. Static Shortest Path Routing Data Structures

4.3.3 Simulation Experiments

The shortest path routing algorithm was simulated for 1000 time units with the source rates shown in Fig. 2B. The switching delay was assumed to be 0.1 time units. The response time for all messages generated by all hosts in the network was measured as the time to travel from msource to dstat. Fig. 3D shows the average network response time histogram calculated by PAWS.

RESPONSES FROM /MSOURCE (1) TO /DSTAT (1)
 CATEGORY: /DATA

RESPONSE-TIME INTERVAL	NUMBER IN INTERVAL	Z IN INTERVAL	HISTOGRAM
			0 10 20 30 40 50 60 70 80 90 100
0.000 <= X < 2.000	125.000	49.80	*****< I
2.000 <= X < 4.000	66.000	26.29	*****< I
4.000 <= X < 6.000	35.000	13.94	*****< I
6.000 <= X < 8.000	12.000	4.78	***< I
8.000 <= X < *INFINITY*	13.000	5.18	**< I
.....			
TOTAL: 251.000			

SUMMARY

MEAN:	2.821	2ND MOMENT:	14.080
VAR:	6.120	STNDRD DEV:	2.474

Fig. 3D Response Time Statistics for Shortest Path Routing

4.3.4 Variations on Shortest Path Routing

The routing algorithm discussed above uses only one route per source-destination pair, which has been pre-computed and fed as input to the FORTRAN program (statrout.for). In general there may be more than one route with the same minimum cost between source-destination pairs. In addition it may be desirable in high traffic situations to distribute some traffic to links that are lightly loaded, or unloaded, in order to improve performance (e.g. in the example network the links between nodes 1 and 3, 3 and 4, and 2 and 6 do not lie on any shortest path and hence are unused). Finally, multiple source-destination paths are needed in case link failures can occur.

One approach to distributing the traffic over several routes for each source-destination pair is to choose one of several routes on the basis of a probability assignment (which can be considered a "load factor") for each route. Thus routes can be ranked in order of preference. This approach has in fact been implemented in the FORTRAN program written for this project. Thus the routing matrix for each switch can contain probabilities assigned to various switches that an incoming message can be forwarded to in order to proceed to the destination. This approach was not pursued further as it was clearly within the capabilities of PAWS and no new insights were expected.

A more sophisticated version of shortest path routing would combine it with some form of adaptive routing, e.g. by considering not only static quantities such as link delay and hop counts, but information that is dynamically available e.g., queue lengths at outgoing lines. This could result in a very effective algorithm that would continue the advantages of shortest path routing with an algorithm such as Hot Potato, discussed in sec. 4.4.

4.3.5 Discussion

One issue associated with shortest path routing is the computation of the shortest paths. In the current model the shortest paths are computed by the model user by inspection. For a large network it would be useful to have a (separate) utility that would take a network connectivity and link delay matrix input and apply, say, Dijkstra's algorithm [SCH 87] to produce the shortest paths. Such a program could conceivably produce multiple paths, paths selected by several criteria (e.g., minimum delay, minimum hop, etc.), or by a combination of criteria.

Another issue that arises is the specification of source rates. This is a vector of length equal to the number of nodes in the network. Unlike the traffic and connectivity matrices however, it cannot be read in from a data input file, but must be specified in the PAWS/GPSM model itself. This is somewhat tedious, and if done in GPSM, requires that the model be retranslated to PAWS for each simulation run in which only the source rates vector is changed.

These issues will be discussed in the enhancement specification.

4.4 Distributed Adaptive Routing

The problem with static directory routing is that it does not adapt to changes in network traffic. One way to overcome this is to periodically recalculate the routing tables based on current estimates of traffic. A centralized approach to adaptive routing usually involves each switch periodically sending its status information (queue lengths, links known to be down, etc.) to a central routing control center, which recalculates the routing tables and redistributes them to the switches. This approach has several drawbacks: the cost and complexity of re-distributing routing tables, the vulnerability of the control center, and the traffic congestion at the center. It is preferable to adapt to traffic changes using distributed control.

A class of algorithms known as isolated adaptive algorithms [TAN 81] attempts to let switches make routing decisions based only on information that they themselves have gleaned. They do not exchange routing information with other switches, so they are often simple and cheap to implement; however, they are less likely to produce optimal routes than true distributed algorithms. Examples of the latter include the distributed algorithms of Jaffe and Moss [JAF 82] and the work of Tajibnapis [TAJ 77].

4.4.1 Hot Potato Routing Model

A simple example of an adaptive routing algorithm with decentralized control is called hot potato [TAN 81]. The idea is simple -- when a message arrives, the switch sends it on the outgoing line with the shortest queue, without regard to the message destination. Clearly this algorithm will not perform nearly as well as static shortest path routing. It can be combined with static routing by making the path metric a function of both link delay and queue length. We have implemented the simple hot potato algorithm for our network example, using queue length information that was conveniently available in the PAWS model.

The switch model for hot potato routing (Fig. 4A) is essentially identical to that for shortest path routing . The only difference is that the PAWS user node croute has been replaced by the COMPUTE node calcroute, which performs the necessary routing calculations using the PAWS computation syntax and not by invoking a FORTRAN program.

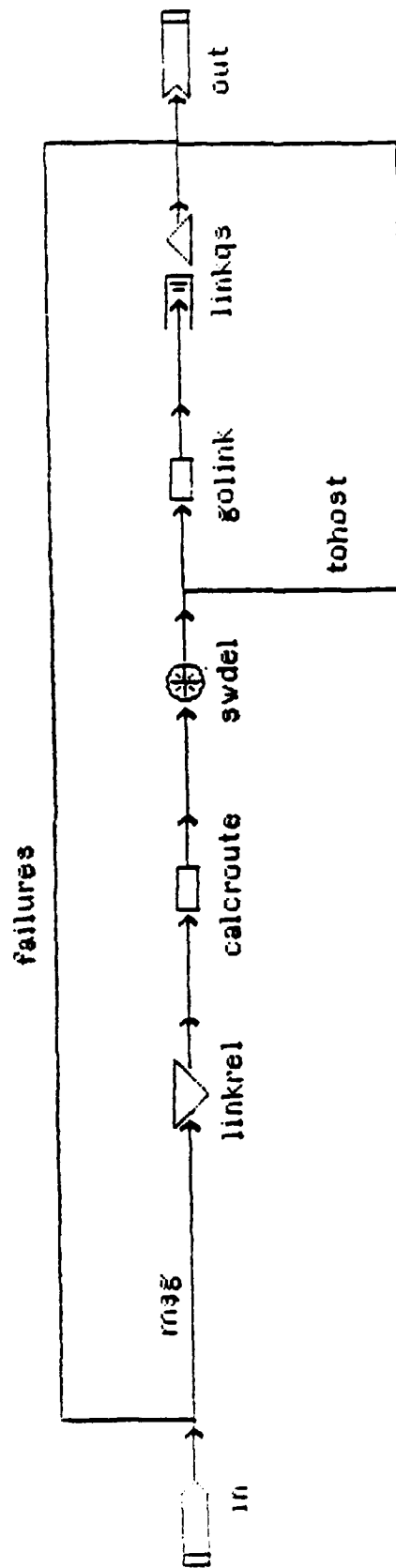


Fig. 4A Hot Potato Routing: Switch Model.

The routing calculation uses several PAWS local and global variables in addition to the ones used for shortest path routing. These are shown in Fig. 4B.

PAWS local integers

LI [1]	message source
LI [2]	message destination
LI [3]	current network switch
LI [4]	next network switch
LI [5]	link delay to be experienced
LI [6]	link number to be travelled
	$= (LI [3] - 1) * 6 + LI [4]$

PAWS local booleans

LB [1]	route message to link
LB [2]	message has reached destination
LB [3]	message loss

PAWS global integers

minql	minimum queue length
minq	link with shortest queue
mini	switch corresponding to minq

PAWS global booleans

GB [1]	no link between some two switches
GB [2]	minimum queue length found

Fig. 4B Significant Data Structures for Hot Potato Routing

When a message arrives at calcroute, the algorithm first checks to see if the destination has been reached. If so, the message is treated as it would be for the shortest path algorithm, and eventually leaves the model via arc tohost. If not, the algorithm computes the link

number for each possible outgoing link from this node. If such a link exists in the example network, the queue length at linkqs for that link is read. This is easy to do because PAWS COMPUTE nodes allow transactions to access queue lengths at any SERVICE, ALLOCATE or GETMEM node within the same submodel. The algorithm finds the outgoing link with the minimum queue length, and assigns the corresponding "next switch" to the message.

4.4.2 Simulation Results

The hot potato routing algorithm was simulated for 1000 time units with a switch delay of 0.1 units. Average response time statistics for all messages for all source destination pairs were obtained by specifying msource and dstat as the PAWS response time from- and to-nodes. The response time histogram calculated by PAWS is shown in Fig. 4C.

```

RESPONSES FROM      /MSOURCE ( 1) TO      /DSTAT ( 1)
CATEGORY:           /DATA
RESPONSE-TIME
INTERVAL              NUMBER IN      Z IN
                      INTERVAL        INTERVAL
0.000 <= X < 10.000      77.000      37.20
10.000 <= X < 20.000      41.000      19.81
20.000 <= X < 30.000      22.000      10.63
30.000 <= X < 40.000      11.000       5.31
40.000 <= X < *INFINITY*   56.000      27.05

HISTOGRAM
0  10  20  30  40  50  60  70  80  90 100
*****<
*****<
*****<
***<
***<
*****<

TOTAL: *****
      207.000
    
```

```

SUMMARY
MEAN:    52.251 2ND MOMENT: 13335.629
VAR:    10605.468 STNDRD DEV: 102.983
    
```

Fig. 4C Response Time Statistics for Hot Potato
Routing Algorithm

Clearly, static shortest path routing is far superior to hot potato routing as the hot potato algorithm does not consider the ultimate destination of the message.

4.4.3 Discussion

The hot potato routing algorithm is an example of how PAWS facilities can be used conveniently for constructing communications models. In this case the queue length information available from PAWS made modeling of this algorithm very easy. The model did not require the user to write and debug a FORTRAN program. Instead the high-level declarative features of the PAWS language were used. In the ES/PAWS tool under development, not only queue lengths but most model parameters will be available to the model developer.

This illustrates two approaches to performing calculations within PAWS models. The first is the approach taken in shortest path routing, which is to do all but the most rudimentary calculations using FORTRAN programs invoked from PAWS USER nodes. The advantages of this approach are:

- 1) the program can be tested in isolation from the simulation model
- 2) data can be read from and written to FORTRAN data files without having to retranslate the model. This provides a degree of isolation between algorithms and data structures, as data need not be embedded in the model.
- 3) the user can use a favorite programming language other than FORTRAN, e.g., Pascal or C.
- 4) complex algorithms can be developed using powerful programming languages
- 5) the resulting model may be more efficient

The second approach is to use as few USER nodes as possible, and to perform all calculations within PAWS COMPUTE nodes. This requires that model inputs be kept in PAWS variables and initialized in the "RUN" section. In fact, the hot potato routing model developed here does just that. The link delays are kept in an array of global reals of size N^2 , where N is the number of nodes in the network. This information is used in the routing algorithm to determine the outgoing links from a switch, and their link delay. It would be possible to take this one step further and store the traffic matrix in this way and

replace the traffic USER node in the main model with a COMPUTE node. The advantages of doing this are:

- 1) the model is written entirely in the PAWS language; the entire model can be viewed using GPSM and so is easier to understand
- 2) the user need not have any knowledge of FORTRAN programming--the high-level PAWS constructs can be used exclusively
- 3) Model parameters, e.g., queue lengths, are available directly

These issues are being addressed by the ES/PAWS software under development. They will be discussed in the enhancement specification.

4.5 Flooding

A simple algorithm that is attractive due to its high reliability is flooding. In one version of flooding, which we shall call naive flooding, every incoming message is sent out on every outgoing line, including the one it arrived on. The message sent back to the switch from which it just arrived is treated as an acknowledgement at that switch. The version of flooding implemented in this project is standard flooding, in which incoming messages are sent out on every outgoing link except the one they arrived on. A variation of this is called selective flooding [TAN 81] in which incoming messages are sent out only on those lines going approximately in the right direction. In all three versions of flooding, various measures can be taken to damp the large numbers of duplicate packets that will be produced, e.g. by putting hop counts in the message, so that once a message has traveled a distance greater than the network diameter it is discarded. Flooding always chooses the shortest path, since it tries every path.

4.5.1 Routing model

In our implementation of flooding every message generated in the model is assigned a unique sequence number. This is so that copies of messages that have already been seen by a switch can be discarded. The sequence number is assigned in the PAWS node makeone in the main model, which has been changed from a simple CHANGE node to a COMPUTE node (see Fig. 5A).

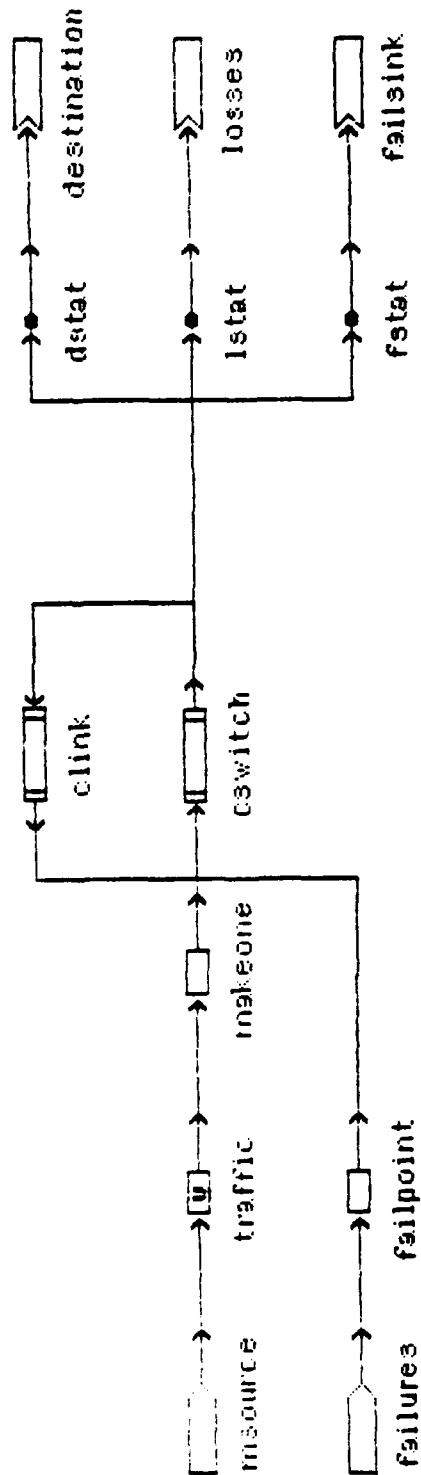


Fig. 5A Flooding: Modified Network Model.

Although the flooding algorithm is conceptually quite simple, the performance model is a little tricky. The main problem is to ensure that the model discards exactly the right number of duplicates. This is discussed in sec. 4.5.2 below.

For the moment assume that duplicate messages are correctly generated and discarded, and that this calculation is carried out in the PAWS nodes calcroute and flood of the flooding switch model (Fig. 5B). The node calcroute determines whether an incoming message has reached its destination, in which case it eventually leaves the model via arc reached, or whether the incoming message should be flooded out on the outgoing links, in which case it proceeds to makecopy.

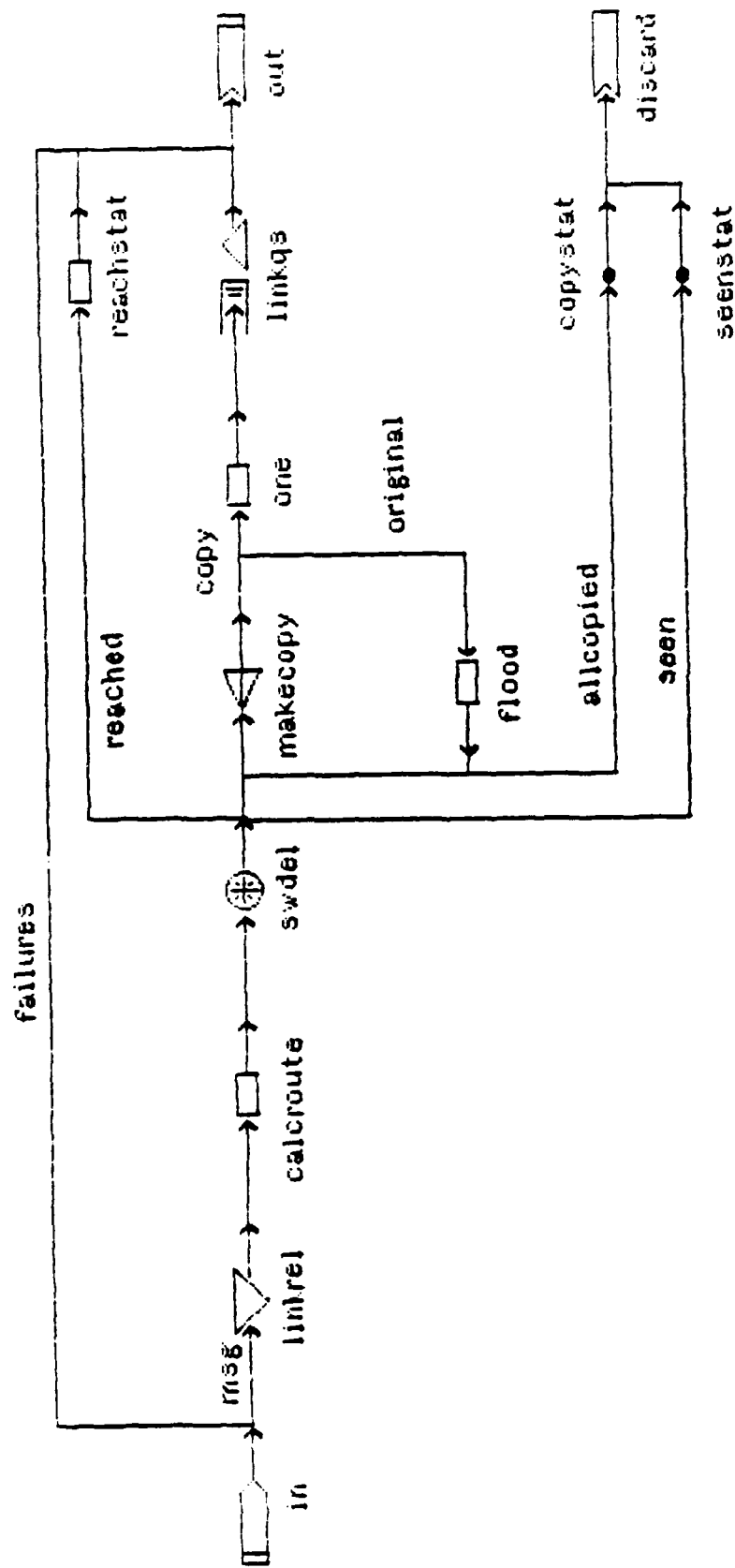


Fig. 5B Flooding: Switch Model.

The first time the message reaches makecopy it creates zero copies (PAWS siblings), and proceeds to node flood. Here the second part of the flooding algorithm – copy generation – is carried out. The original message arriving at flood leaves with the link number of the link on which a copy should be sent. (The link number is determined in a straightforward way so that a copy is not sent on the link on which the message arrived, and no attempt is made to send on non-existent links). The original message circulates between makecopy and flood creating one new copy each time it reaches makecopy. The copies leave the loop and proceed for processing to node one, while the original leaves the model via arc allcopied when all the necessary copies have been created. Copies undergo some minor "book-keeping" changes at node one before competing for the appropriate outgoing link at links. The link allocation and release is handled exactly as for the shortest path routing model.

4.5.2 Discarding Duplicates

The generation of copies is quite straightforward, but not the discarding of duplicates. One attempt to do this is to keep an array of local flags in each message, one per switch. When switch J receives a message it checks to see if flag J is set in the message. If so the message has already been seen by this switch, and is discarded. If not, the flag is set so that any future duplicates will be recognized, and the message is forwarded for the creation of copies. (When PAWS creates copies of transactions at a SPLIT node, the local flags of the original transaction are replicated in the copies). Unfortunately, although this scheme seems intuitively clear, it is not correct. To see this consider the network example. A message generated at host 1 will be copied and sent to switches 2, 3, and 4. Say the message routed via switch 2 reaches switch 3 before any of the others do. This message will have flags 1 and 2 set. Now consider that the message routed via switch 4 arrives, with flags 1 and 4 set. It will not be recognized as a duplicate.

The problem is that marking messages in this way only ensures that copies that each switch has itself generated are recognized as duplicates when they circulate through the network and arrive at the same switch again. It does not ensure that copies that have taken different paths to the switch are recognized as duplicates.

One aim of discarding duplicates is to ensure that the destination host only actually receives one copy of every message destined for it. An attempt to do this is to abandon our first scheme and instead to generate unique sequence numbers for each message. In this second

scheme each switch keeps in an array of global variables the last sequence number seen for each destination. At first it may seem that one would need to keep a list of all the sequence numbers seen by a given switch. However, in the absence of message failures, it would seem sufficient to keep only the last one. The argument is to consider messages generated by host 1 for host 6 in the example network. Say messages with sequence numbers x and y are generated, in that order. In that case message y follows message x on every outgoing link. If there are no failures, the messages will reach switches 2, 3, and 4 in this order, and since switches do not re-order messages, will reach host 6 in the correct order. So it appears to be sufficient for a host to keep the sequence number of the last message received, and discard any message received with smaller or the same sequence numbers.

Notice that this scheme preserves end-to-end duplication control: no destination host will receive duplicate messages, and in that sense the scheme seems correct. However the scheme does not meet the stronger requirement of flooding, i.e., that no switch in the network forwards a message that it has seen already. The reason is that at intermediate switches in the network the message is not forwarded to the host, so duplicates will not be recognized until they eventually reach the destination switch. A lot of unnecessary traffic will be generated and the network will eventually overflow. One might think that a way to overcome this would be to combine schemes 1 and 2, so that intermediate switches would recognize duplicates. This third scheme will work in the sense that no message duplicates will circulate in the network forever. However, intermediate switches would only recognize duplicates that they had seen before, and not those that took different paths to arrive there. Hence scheme 3 wastes the network capacity, so that at high loads the network will overflow for loads which it should technically be able to sustain. In fact when scheme 3 was implemented for the example network with the input rates shown in Fig. 2B, the simulated network did overflow.

The following scheme is actually the scheme that was implemented for this project. Each switch maintains a vector containing the last sequence number it received from each source host. Messages seen from a source with smaller or equal sequence numbers are discarded. Let us see why scheme 4 works.

First note that the argument given in scheme 2 for needing to keep only the last sequence number, instead of a list, is only correct if one merely considers messages between a single source-destination pair. Although messages from a single source arrive at a given destination in order (by the argument given in scheme 2), it is entirely possible that

messages generated later in time (and hence having a greater sequence number) at an adjacent node will arrive earlier, so that messages from distant nodes would be discarded incorrectly. Thus scheme 2 fails not only because intermediate switches do not recognize duplicates, but because some messages may be discarded incorrectly. The correct way to uniquely identify duplicates is by keeping the sequence number of the last message seen from a given source host.

Now consider what happens if two copies of a message arrive at a switch by following different paths. The one that reaches first will have its sequence number entered in the slot for this source host, and will be flooded on outgoing links. The second message to arrive will be recognized as a duplicate and will correctly be discarded.

4.5.3 Network Overflow

The network overflows when the traffic input to the network is greater than its processing capacity. In our example the switching delays are assumed to be small, so the bottleneck is the link delay. In standard flood routing with an incorrect implementation (scheme 3) the example network was observed to overflow, and queue lengths at some of the links grew without bound (at PAWS node links).

It is useful to estimate whether network overflow will occur prior to running any simulation experiments. For flood routing a simple bottleneck analysis is possible. Instead of considering standard flooding, suppose naive flooding was used. Naive flooding will clearly generate more network traffic than standard flooding, but is easier to analyze, and provides a conservative estimate of the bottlenecks. Under naive flooding each message generated by a source node will traverse every link twice. If M messages are generated per time unit, each link will eventually be traversed by M messages in each direction. If the link capacity of the slowest link is C_{\min} in each direction, network overflow will occur unless

$$M < C_{\min}$$

In our example messages are generated with inter-arrival times of 10, 20, 30, 40, 50 and 60 time units. This corresponds to an average aggregate arrival rate of

$$M = 0.245 \text{ messages/time unit}$$

The slowest link has a delay of 5 units per message, i.e.,

$$C_{\min} = 0.2 \quad \text{messages/time unit}$$

Thus under naive flooding the network will overflow. The actual behavior under standard flooding is hard to analyze, but it is likely that the network will not overflow since M is relatively close to C_{\min} . This turns out to be the case.

4.5.4 Simulation Results

The flooding algorithm was simulated for 1000 time units. Under the current implementation of the model it is not possible to obtain response time histograms. This is because PAWS generates response time statistics for each original message. However, in this model the original message is lost once it has generated all the copies it needs to at the first switch. For this reason it was necessary to calculate the travel time for each message, including its copies and calculate the net travel time from msource to the PAWS node reached. This was divided by the throughput count at node reached. This yielded a mean response time of 2.67 units.

4.5.5 Discussion

The flooding algorithm raises several interesting points. An obvious consideration is that due to the way response time statistics are collected in PAWS it is not possible to obtain a response time histogram for the model as it currently stands. A simple change to the model to overcome this would be to insert a SPLIT and an ALLOCATE node between msource and cswitch. One sibling is created at the SPLIT which traverses the network, while the original message waits at the ALLOCATE. The sibling carries the ID of the waiting transaction; when it (or a copy) reaches the destination, it interrupts the waiting transaction, which proceeds to a sink. The response time for the message traversal through the network can be found as the response time of the waiting transaction. Since this idea was clearly within the current capabilities of PAWS it was not implemented.

A more interesting issue is raised by the complexity of modeling flooding correctly, as discussed in sec. 4.5.2. This points out that there needs to be a clear distinction in C/PAWS between model developers, who would deal with the problems discussed in sec. 4.5.2, and model users, who would be more interested in varying the parameters and

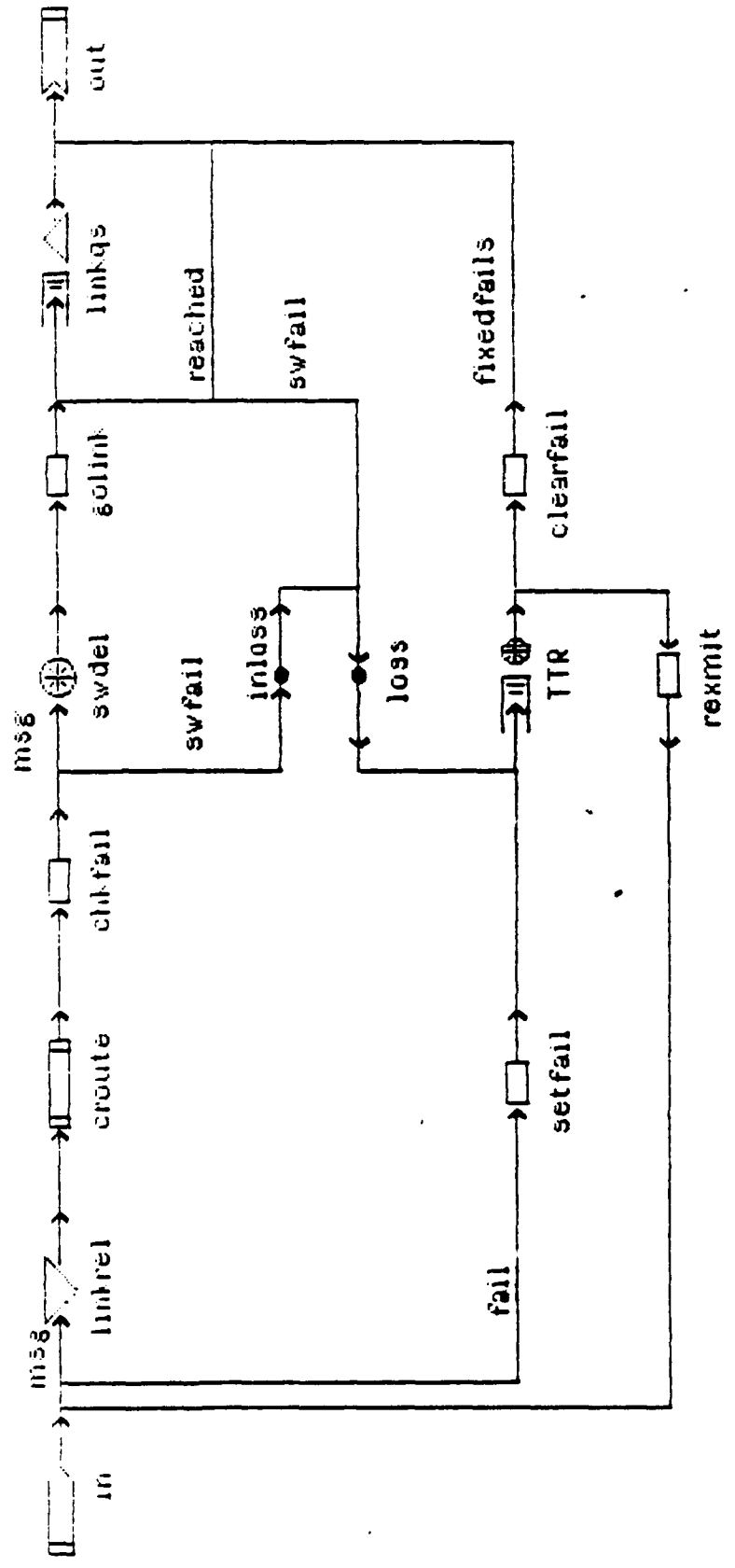
inputs to the model for design or performance studies. IRA is currently seriously studying the distinction. Another interesting point is that it is desirable to assist the model developer in gaining confidence that the model is correct. This aspect of performance modeling is often neglected [LAW 82]. This issue will be discussed in the enhancement specification.

4.6 Shortest Path Routing With Failures

All the previous models assume there are no failures in the network switches or links. In this model the effect of failures on static shortest path routing is studied. Failure modeling has been studied fairly extensively by IRA [FER 84, PAL 85, VEL 86] and IRA clients. In this project it was decided merely to demonstrate that failures can be modeled quite easily for the example problem chosen here.

4.6.1 Failure Model

Failures are generated at the node failure in the main model with an exponential distribution with a mean specified by the PAWS global variable Mtbf (see Fig. 2A). At node failpoint in the main model each failure transaction is assigned the switch (or link) that it is going to cause to fail. To simplify the following description assume that only switches can fail; links are handled similarly. It is assumed that all switches fail equally often. Assume switch *i* has failed. The failure transaction then enters submodel switch (see Fig. 6A) and proceeds to compute node serfail, where it sets a global flag indicating that switch *i* has failed. The failure transaction then waits at TTR (*i*) for a time with an exponential distribution with mean Mttr, which represents the mean time to repair the switch. At the end of this time the failure transaction clears the failure flag (clearfail) and leaves the model.



Data messages that arrive at switch i undergo the usual shortest path routing calculation (calcroute), and check to see if the current switch has failed at node chkfail (The switch may also fail while the message is being processed. This is checked at node golink). If so, the message proceeds to the i th service node TTR (i), where it waits in the queue behind the failure transaction that caused switch i to fail. Note that in this way messages that arrive in the middle of a switch repair only wait for the remaining time to repair the switch. After the failure transaction leaves, the message waits at TTR (i) for a time equal to the mean time to retransmit the message, Mttxmit. (It is assumed that some other network layer is responsible for actually generating retransmissions. In this model the performance impact of retransmissions is modeled by this extra processing delay). There is a slight inaccuracy in the model here in that a new failure arriving at switch i would have to wait for retransmissions to complete before bringing the switch down. However, since Mttxmit is much less than Mtbf, and the probability of the same switch failing twice in a row is low, this scenario is ignored. The data messages are reinitialized at rexmit before being fed back into this switch at node linkrel.

The failure of links can be modeled in a completely analogous way. Since this is clearly within the capabilities of PAWS, and since failure modeling has already been studied so extensively by IRA, link failure was not modeled for this project, as it was not felt that any new insights were to be gained by doing so.

4.6.2 Simulation Experiments

The model was run for 1000 time units using exactly the same parameters as in sec. 4.3.3. The following values were used:

Mtbf	100	time units
Mttr	25	time units
Mttxmit	3	time units

The average response time distribution histogram computed by PAWS is shown in Fig. 6B.

```

RESPONSES FROM      /MSOURCE ( 1) TO      /DSTAT ( 1)
CATEGORY:           /DATA
RESPONSE-TIME
INTERVAL              NUMBER IN          % IN          HISTOGRAM
                        INTERVAL          INTERVAL
0.000 <= X <      2.000      97.000      39.55      I*****<
2.000 <= X <      4.000      76.000      34.55      I*****<
4.000 <= X <      6.000      30.000      13.64      I*****<
6.000 <= X <      8.000      13.000       5.91      I**<
8.000 <= X < *INFINITY*  14.000       6.36      I**<
                        .....
TOTAL:      220.000

```

```

SUMMARY
MEAN:      3.603 2ND MOMENT:      58.801
VAR:      45.819 STNDRD DEV:      6.769

```

Fig. 6B Response Time Statistics for Shortest Path
Routing with Failures

4.6.3 Adaptive Shortest Path Routing

The model discussed in sec. 4.6.1 considers the effect of failures when a static shortest path routing algorithm is used. We now consider an adaptive shortest path algorithm which bypasses a failed switch. The GPSM graph for the switch submodel of this new model is shown in Fig. 6C.

Consider switch K in the network. When there are no failures, messages proceed through the switch exactly as described in sec. 4.6.1. However, the switch behavior is more complex when failures (and repairs) occur.

When switch K fails

- 1) host K is isolated from the rest of the network. Messages from host K (new or those already in process) proceed via arcs localfail to wait at node TTR for the failure to be repaired.
- 2) messages from an adjacent switch J at switch K (i.e., those already in process) would, in the real network, be lost. In the model they proceed via arcs remotefail to wait at seexmit for a time that includes:
 - a) time for switch J to detect the failure, and
 - b) time for switch J to retransmit the message.

The messages are modified at lastswitch so that when they reach croute they appear to have just arrived at switch J, which, having detected that switch K has failed, assigns them an alternate route that bypasses switch K. This routing calculation can be done using the sophisticated loop-free routing algorithm described in [JAF 82].

- 3) new messages arriving at switch J that would have been routed through K are also assigned the alternate route.

When switch K is repaired:

- 1) host K messages wait at TTR a further time Mttrexmit before proceeding through node rexmit back into switch K
- 2) adjacent switches revert to routing messages via switch K.

This algorithm adapts to the failure of switches. The model realistically captures the following effects of a switch failure.

- a) the increase in message delays
- b) the increase in traffic at adjacent switches

- c) the isolation of a host from a network when its IMP (switch) fails, and the long delays that result
- d) a local error control strategy, where re-routing decisions are made by intermediate switches rather than by the source host

Notice that this model assumes there is no delay between a switch failure or repair occurring and this occurrence being detected by other switches in the network. In that sense, this model does not consider the transient effects of failures.

This points out an interesting aspect of failure modeling. The system behavior when a failure occurs can be characterized as consisting of three phases. In the first phase, the component has failed but adjacent components are not aware of the failure. They will experience a timeout while waiting for an acknowledgement, and will retransmit a copy of the message via the original route, i.e., to the failed component. In phase 2 (some of) the adjacent components will become aware of the failure, and will propagate this information to the rest of the network. Depending on the error detection scheme used, this phase may or may not take substantial time and involve generation of additional system messages. In phase 3, all active components of the network are aware of the failure and messages are re-routed as appropriate. In phase 3 a form of "failure steady state" has been reached while the first two phases represent transient response.

It is expected that the performance degradation in phase 3 is due mainly to operating the system with depleted resources. Thus messages may have to take longer routes to reach their destinations, and there will be additional congestion at network switches. In phase 1, on the other hand, performance degradation is due mainly to the timeout and retransmission of messages. This will increase link and switch traffic. In the scheme described here, where error control is carried out between adjacent switches, there will be increased traffic between the failed component and adjacent switches. The performance degradation in phase 2 may include the effects of increased traffic due to retransmissions for some switches, longer delays due to depleted resources for switches aware of the failure, and additional traffic due to system messages that propagate information of the failure. However, depending on the error detection scheme used, phase 2 may be short-lived, and the additional system traffic may be small.

The model constructed for static routing in sec. 4.6.1 can be compared with the adaptive routing model in this light. The former model corresponds to phase 1 of system behavior when the system adapts to failures, while the latter model corresponds to phase 3. In that sense the model of sec. 4.6.1 is more conservative than the model discussed here, and can be used to determine whether a failure would cause unacceptably poor performance or violate some network real-time constraint.

It is clear from the switch model in Fig. 6C that adaptive routing can be modeled using PAWS; in fact the GPSM graph itself only requires some minor modifications. For this reason this model was not actually constructed, as it appeared that it was clearly within the capabilities of PAWS.

4.6.4 Discussion

Failure modeling has been discussed at length in [FER 84, PAL 85, VEL 86] and also in various proprietary technical reports prepared by IRA clients. Two points that can be gleaned from the experience gained in this project are discussed briefly here. The first is that this model treats the occurrence of failures (and repairs) as a process independent of the occurrence of data messages. This seems to be more realistic than simply causing data messages to fail with a certain probability because:

- 1) in the latter case one should use the conditional probability of failure, i.e., the probability of failure given that a message has been generated.
- 2) failure rate measurements are often done by measuring the occurrence of failures as continuous bit streams traverse a component. This provides an estimate of the unconditional probability of failure.

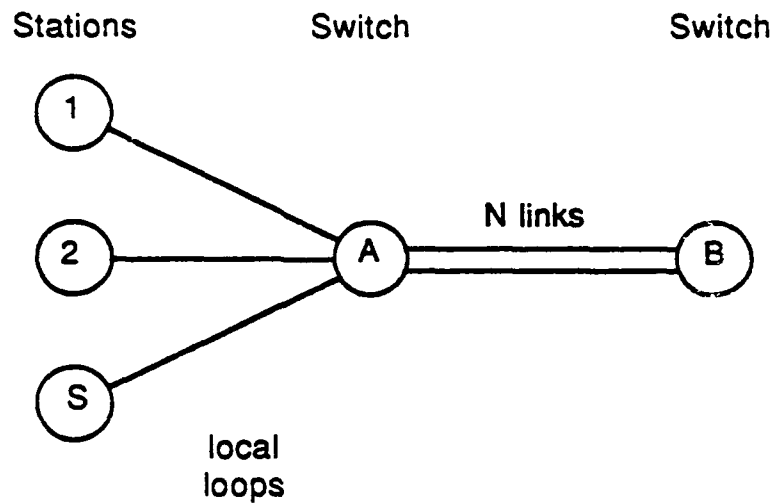
The second point raised by this model is the possibility of introducing some high-level facilities for modeling failures in C/PAWS. This is discussed in the enhancement specification.

5.0 Communications Switch Model

The switching element model studies the behavior of a switch in a circuit-switched network. The switch receives call requests from stations connected to it, and attempts to assign outgoing lines to satisfy these requests. Eventually all outgoing lines are allocated, and further incoming requests are called "lost calls". Circuit switches either discard these requests (lost calls blocked) or queue them internally (lost calls delayed) [HAM 86]. The performance metric of interest is the blocking probability in the former case and the connect time in the latter. This model estimates connect time for a simple queued mode (lost calls delayed) circuit switch.

Since the object of the model is not to study routing, it is assumed that the switch is directly connected to every switch in the network. In particular, there are N links between switch A, the switch of interest, and switch B. There are S stations which forward call requests to switch A via local loops, dedicated links between the station and the switch. The call processing scenario is also shown in Fig. 7A.

Network:



Call Processing:

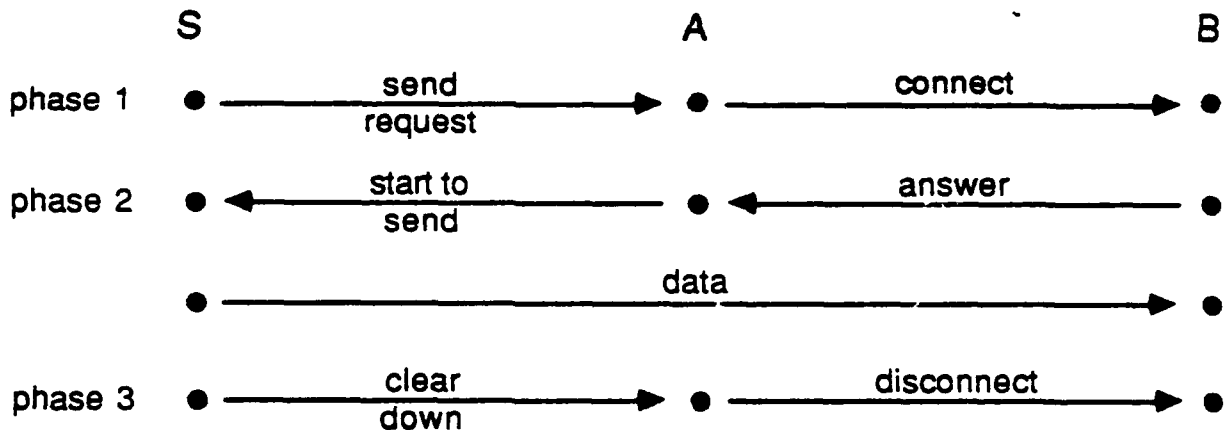


Fig. 7A. Call Processing Scenario

A station S sends a send request message to switch A in an attempt to connect with switch B. When a line becomes available, a connect message is forwarded to switch B, which responds with an answer message. Switch A then sends a start-to-send message to S, which begins the actual data (or voice or text) transmission. At the end of the transaction S sends a clear down message to A, which sends a disconnect to B. The ringing and response at switch B's station is ignored [SCH 87].

The connect time for a message from station S is the time from when station S sends a send request to the time it receives a start-to-send. The hold time for a call is the amount of time a link is held for that call, and equals the time from the connect message being sent from A until the disconnect message is processed at B.

5.1 Switch Model

Each of the control messages is modeled as a constant time delay in the model (nodes sendreq, connect, answer, startsend, cleardown, disconnect), shown in Fig. 7B.

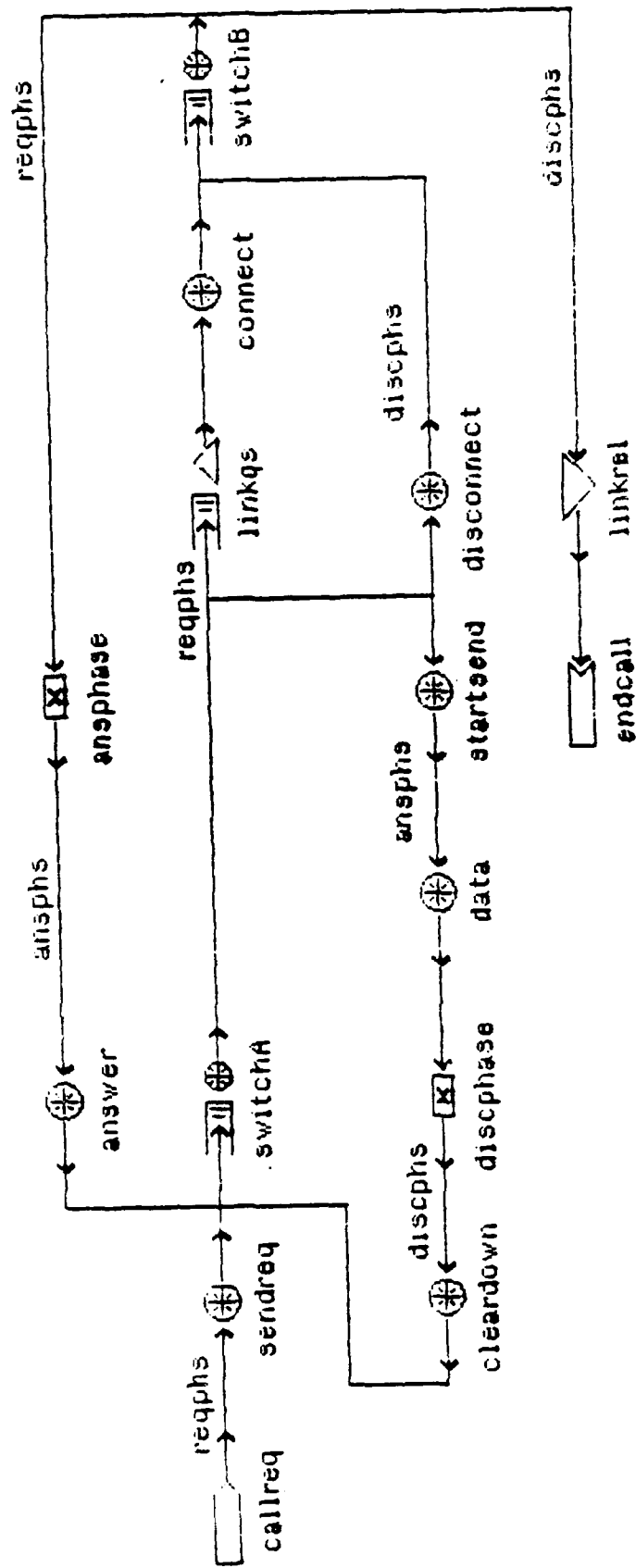


Fig. 7B Communications Switch Model

The three phases of call processing are represented by arcs reqphs, ansphs, and disphs, and correspond to PAWS transaction phases 1, 2 and 3. The single server queues switchA and switchB represent the contention for processing at each switch. The single ALLOCATE node linkqs contains N tokens -- one per outgoing link. Lost calls are queued at linkqs. Links are released at the end of a call at node linkrel, while the duration of a call itself is assumed to have an exponential distribution, modeled a node data.

Calls in the request phase go from callreq to switchB where they wait. After processing at B the calls are now in the answer phase, and circulate via answer and switchA to startsend. Calls that are in the disconnect phase leave data and proceed by nodes switchA and disconnect to switchB, and eventually leave the model via endcall.

5.2 Simulation Experiments

The queued node circuit switch was simulated for 10,000 time units using the same parameters as in [SCH 87]. These are shown in Fig. 7C, along with a histogram of the average connect time, i.e., the response time for messages going from callreq to startsend. Messages were generated with an exponentially distributed inter-arrival time Trequests. All control messages were assumed to have the same constant delay (Tsendreq = Tanswer = Tstartsend = TCD = Tdisconnect = 0.091) except for the connect message, which was assumed to have a much larger delay as it carries substantially more information (Tconnect = 0.91). Data messages were assumed to experience a delay Tdata exponentially distributed with a mean of 9.1 units. The processing time of the switches was very close to zero. With these assumptions the average connect time was found to be 1.76 units, which agrees closely with the value obtained from an analytic model: 1.7 units [SCH 87].

RESPONSES FROM /CALLREQ (1) TO /STARTSEND (1)
 CATEGORY: /REQ

RESPONSE-TIME INTERVAL		NUMBER IN INTERVAL	% IN INTERVAL	HISTOGRAM
				0 10 20 30 40 50 60 70 80 90 100
0.000 <= X <	0.500	0.000	0.00	I< I
0.500 <= X <	1.000	0.000	0.00	I< I
1.000 <= X <	1.500	5466.000	82.27	I<*****I
1.500 <= X <	2.000	195.000	2.93	I< I
2.000 <= X < *INFINITY*		983.000	14.80	I<*****I
TOTAL:		6644.000		

SUMMARY

MEAN: 1.757 2ND MOMENT: 5.745
 VAR: 2.659 STNDRD DEV: 1.631

VALUES OF SCALARS:

/NUMLINKS = 10
 /TREQUESTS = 1.500
 /TSENDREQ = 0.071
 /TCONNECT = 0.910
 /TANSWER = 0.091
 /TDATA = 9.100
 /TCD = 0.091
 /TDATACD = 9.191
 /TSTARTSEND = 0.091
 /TDISCONNEC = 0.091
 /TSWITCHA = 0.010
 /TSWITCHB = 0.010

Fig. 7C Connect Time Statistics for Queued Mode
 Circuit Switch

5.3 Discussion

This model is a simple but fairly useful simulation of queued mode call processing for communication between two network switches. It is quite easy to generalize to the case where there are M switches with a different number of links between switch A and other switches by using arrays of SERVICE and ALLOCATE model. Since this can clearly be done within PAWS it was not pursued.

An interesting point is that to be accurate in the case where there is more than one switch to which switch A is connected, the "background" traffic in the network needs to be simulated. The background traffic represents the load at switchA and switchB due to call processing for calls between switch A and other switches, and switch B and other switches. One approach would be to model this traffic precisely using arrays of SOURCE and ALLOCATE nodes, one per source-destination pair. There is a better approach. Since the call processing scenario is deterministic, the background processing can be calculated quite accurately for a certain network node, and fed as transactions that interfere with A-B communication at nodes switchA and switchB. This will give a more realistic estimate of network behavior, without the complexity of detailed circulation.

6.0 Conclusions

The modeling and experimental evaluation task of this phase I project has demonstrated that it is feasible to model many aspects of communications systems using PAWS. Several issues were raised during this task, and some limitations were found in PAWS, many of which will be overcome in ES/PAWS. These issues will be discussed in detail in the enhancements specification.

References

AND 84

G. E. Anderson "The Coordinated Use of Five Performance Evaluation Methodologies", Communications of the ACM, Vol. 27, no. 2, 1984.

BRO 88

J. C. Browne, P. Jain, D. M. Neuse and M. Esslinger, "ES/PAWS - A System Level Design Aid," submitted for publication in Proceedings of the Design Automation Conference, June, 1988.

CON 86

Customer confidential document. The model involves a Sperry 1100/44 computer system with multiple command, arithmetic and peripheral processors used for real-time computations. Permission is being sought to disclose this information.

DOR 84

Vladimir Dorfman, "SNA Communication Line Performance Analysis", Fujitsu Systems of America, Technical Report 92-00024, July, 1984 (proprietary).

FER 84

V. Fernandes, J. C. Browne, D. Neuse, and R. Velpuri, "Some Performance Models of Distributed Systems", Proceedings of the CMG XV International Conference, Dec., 1984.

HAM 86

J. L. Hammond and P. J. P. O'Reilly, Performance Analysis of Local Computer Networks, Addison-Wesley, 1986.

IRA 86

"PAWS Performance Models of a Computer Network Hub", Information Research Associates, Internal Document, 1986.

IRA 87a

Information Research Associates, Performance Analyst's Workbench System (PAWS) User's Manual, 1987.

IRA 87b

Information Research Associates, Graphical Programming of Simulation Models (GPSM) User's Manual, 1987.

JAF 82

M. Jaffe and F. H. Moss, "A Responsive Distributed Routing Algorithm for Computer Networks", IEEE Trans. On Comm., Vol. COM-30 no. 7, pp. 1758-1762, July 1982.

JAI 87

Prem Jain, "Architecture Design of a VLSI Chip Using PAWS/GPSM", Technical Report, Information Research Associates, July 1987.

LAW 82

A. M. Law and W. D. Kelton, Simulation Modeling and Analysis, McGraw-Hill, 1982

NAV 87

United States Navy Contract No. N60021-86-C-0145, High-Level Simulation of Electronic Systems, 1987.

PAL 85

Annette Palmer, J. C. Browne, J. Silverman, A. Tripathi, and R. Velpuri, "A Performance Model of a Fault-Tolerant Distributed System for Evaluating Reliability Mechanisms", Proceedings of the CMG XVI International Conference, 1985.

SCH 87

Mischa Schwartz, Telecommunication Networks: Protocols, Modeling and Analysis, Addison-Wesley, May 1987.

TAJ 77

W. D. Tajibnapis, "A Correctness Proof of a Topology Information Maintenance Protocol for Distributed Computer Networks", Comm. ACM, vol. 20, pp. 477-485, July 1977.

TAN 81

Andrew Tanenbaum, Computer Networks, Prentice-Hall, 1981.

UPC 84

E. Upchurch, "Modeling Packet-Switched Interprocessor Communications", Proceedings of the 15th Annual Modeling and Simulation Conference, University of Pittsburgh, 1984.

VEL 86

Rajkumar Velpuri, "Performance Study of Zeus Distributed System with Different Communication Networks", M. S. Thesis, The University of Texas at Austin, May, 1986.

Appendix B

C/PAWS Enhancement Specification

Contents

1.0	Introduction	B1
2.0	Summary of Proposed Enhancements.....	B2
3.0	User Interface	B4
3.1	Model User Interface versus Model Developer Interface.....	B4
3.2	Forms and Menus.....	B4
3.3	Graphical Interface.....	B8
3.4	Graphical Output	B10
3.5	Network Topology.....	B10
4.0	Modeling Methodology.....	B11
4.1	Submodels and Libraries.....	B11
4.2	C Function Library	B12
4.3	Integrity Constraints and Reasonableness Checks	B12
4.3.1	Node and Transaction State Integrity	B12
4.3.2	Statistics Integrity	B13
4.3.3	Submodel Integrity	B13
4.3.4	Reasonableness Checks	B13
4.4	Statistics Requests	B14
5.0	Communications Systems Features	B15
5.1	Timeout Processing.....	B15
5.2	Flow Control.....	B18
5.3	Routing Algorithm Utilities.....	B18
5.4	Analytic Bottleneck Analysis.....	B19
5.5	Hierarchical simulation.....	B19
5.5.1	Definition	B20
5.5.2	General Approach	B20
5.5.3	A simple example	B21
6.0	Language Enhancements.....	B22
6.1	Failure Modeling Methodology	B22
6.2	Interrupt Resumption Nodes	B23
6.3	FORK and JOIN Constructs.....	B24
6.4	Source Node Control.....	B24

Contents (Cont'd)

7.0 Conclusions..... B25

References..... B26

CECOM COMMUNICATIONS SYSTEMS MODELING PROJECT PHASE I C/PAWS ENHANCEMENT SPECIFICATION

1.0 Introduction

This report discusses the features and enhancements required in C/PAWS in order to make performance modeling of Army communications systems easier and more efficient. C/PAWS will be based on IRA's existing simulation products PAWS and GPSM [IRA 87a, IRA 87b], and more directly on ES/PAWS [BRO 88], a top-down design tool for electronic systems being developed for the Navy.

The object of this Phase I study is to investigate the feasibility of using a simulation tool based on PAWS and GPSM to model the Army's communications systems. This Phase I study was accomplished through discussions with CECOM personnel, study of the CECOM communications systems literature, constructing prototype models of some representative Army communications systems, protocols and algorithms [IRA 88], and by drawing upon IRA's previous experience in this area [FER 84, PAL 85, VEL 86]. This enhancement specification discusses the many issues raised by this investigation, and how they relate to the feasibility of using PAWS/GPSM to model Army communications systems. An attempt has been made to be as specific as possible when discussing these issues while avoiding placing unnecessary constraints on the C/PAWS implementation to be carried out in Phase II. Where possible the high-level issues have been discussed in detail, and some alternative implementation strategies suggested, rather than choosing one implementation technique and specifying it in fine detail. The object is to discuss the feasibility of using PAWS and GPSM, rather than to specify an implementation of C/PAWS. The detailed technical implementation decisions will be the focus of the early stages of Phase II.

The following sections are broadly classified into the enhancement issues relating to the user interface, modeling methodology, language enhancements, and communications systems features. Many of the enhancements desired in C/PAWS will in fact be available with the ES/PAWS system under development. The following section summarizes the enhancements discussed in this report.

2.0 Summary of Proposed Enhancements

ES/PAWS and C/PAWS will be more flexible and powerful than the currently existing PAWS software, and in particular will incorporate the expressiveness of the C programming language. Through the use of the highly user-friendly graphical interface, GPSM, and the use of libraries of submodels and procedures, a user will be able to simulate complex communications systems without needing, in general, to actually write code. Creating and maintaining the submodels and associated procedures, however, will require some familiarity with ES/PAWS syntax; this would be carried out by a model developer rather than a model user (see sec. 3.1).

Using C/PAWS the user will be able to obtain detailed simulation results; these results will then be post-processed by other software packages to display the results of a single simulation run graphically (such as histograms of response times), as well as the results of several simulation runs (such as variations in response time as the offered load is increased upon each new run).

Some features of ES/PAWS particularly useful for communications modeling are:

- access to the C language
(data structures, expressions, functions, etc.)
- macro pre-processor
- submodel parameters
- finite-state machines
- enhanced version of GPSM, including:
 - structured input facility (SIF)
 - graph library facility
 - scrolling of GPSM graphs
 - improved on-line help
 - critique of GPSM graphs

The proposed specific C/PAWS enhancements include:

- 1) Forms and menus for the model user interface
- 2) User-defined icons for specialized components
- 3) Convenient specification of statistics collection, including statistics involving submodels
- 4) Processing of simulation output for graphical display
- 5) A TIMEOUT node type for modeling communications protocol features
- 6) An Interrupt Resume node type for making failure modeling easier. This fits into an overall failure modeling methodology
- 7) Libraries of C functions for analytic bottleneck analysis and routing algorithm calculations
- 8) Libraries of re-usable submodels encapsulating common communications subsystems
- 9) Specification of integrity constraints and reasonableness checks for nodes, transaction states, submodels and simulation results
- 10) More flexible FORK and JOIN constructs
- 11) Explicit control of the generation of transactions at SOURCE nodes, allowing a user to control a simulation run more closely
- 12) Support for hierarchical simulation

3.0 User Interface

GPSM currently provides a highly friendly and intuitive interface to PAWS. The interface to C/PAWS will be made even more powerful and easy to use, as outlined below. Many of the suggested enhancements are currently planned or being implemented as part of ongoing work for ES/PAWS and other IRA projects.

3.1 Model User Interface versus Model Developer Interface

GPSM and ES/PAWS provide an excellent interface for the model developer. A model developer requires knowledge of the PAWS language syntax and some programming skill. The everyday user of a model, however, may be someone such as a communications engineer, network capacity planner, or in general someone other than the model developer. A model user is someone with technical expertise in the area of communications systems who is typically concerned with obtaining simulation results for design or performance evaluation purposes. This is particularly true in the area of communications systems where experts with highly specialized knowledge of communications may not be familiar with programming. Such a user may not wish to see the GPSM or C/PAWS implementations of various communications system components.

This section discusses C/PAWS enhancements that are aimed primarily at a model user. Later sections discuss enhancements aimed primarily at a model developer.

3.2 Forms and Menus

An ideal interface for the model user would consist of standard forms and menus. These could prompt the user for communications systems parameters and configurations. This feature not only provides convenience but also reduces user errors.

The form or menu template itself can be created for a model by the model developer, thus allowing developers (such as Cecom or its consultants) to provide customized model interfaces for their users.. One approach to implement this is to allow C/PAWS to invoke commercially available forms management software package. There are a wide variety of such forms managers available in the environments where C/PAWS will run.

An alternative implementation is to use IRA's EDGE graphical package and SIF (Structured Input Facility) to provide a superior, customized facility. An example of a SIF form for the communications switch model developed for this project is shown in Fig. 1. The user may input switch parameters from a sub-form (Fig. 2), and choose between blocked and delayed calls from a menu. We propose to use EDGE and SIF for this purpose during Phase II.

Communications Switch Model	
Control Message Lengths	
Send Request:	Start Send:
Connect:	Clear Down:
Answer:	Disconnect:
Length of Data Message:	
Switch Parameters:	

Figure 1. Example SIF Form

Communications Switch Model	
Control Message Lengths	
Send Request: <input type="text" value="100"/>	Start Send: <input type="text" value="100"/>
Connect: <input type="text" value="100"/>	Clear Down: <input type="text" value="100"/>
Answer: <input type="text" value="100"/>	Disconnect: <input type="text" value="100"/>

Switch Parameters	
Buffers: <input type="text" value="100"/>	Outgoing Lines: <input type="text" value="100"/>
Lost Calls	<input type="text" value="Delayed"/> <input type="text" value="Blocked"/>

Figure 2. SIF Form with a Sub-Form and a Menu Open

3.3 Graphical Interface

It is desirable to allow the user of the simulation system to create meaningful icons, e.g., representing switches, stations etc., and associate icons with submodels representing those system components. This capability will be provided by applying the submodel class/instance concept in ES/PAWS. A user will be allowed to create a specialized icon to represent a submodel (or node) class. This icon will replace the submodel call node in a GPSM graph. Once created, these application-specific icons for submodels could be applied by all users. Each instance of the user icon will have parameters (instance variables) that can be filled in independently by a user.

The capability for a user to define a specialized icon already exists in the palette editor available with EDGE, the graphics software upon which GPSM is based. Examples of user-defined icons are shown in Fig. 3. The palette editor allows a user not only to draw the icon but specify its graphical characteristics, e.g., the entry and exit points, the flipping operations that are allowed on it, etc. This facility can be extended so that:

- 1) A user can create a specialized icon for a GPSM submodel class that will take the place of the call node for that submodel.
- 2) A user can conveniently specify the parameters of a particular instance of a user-defined icon.

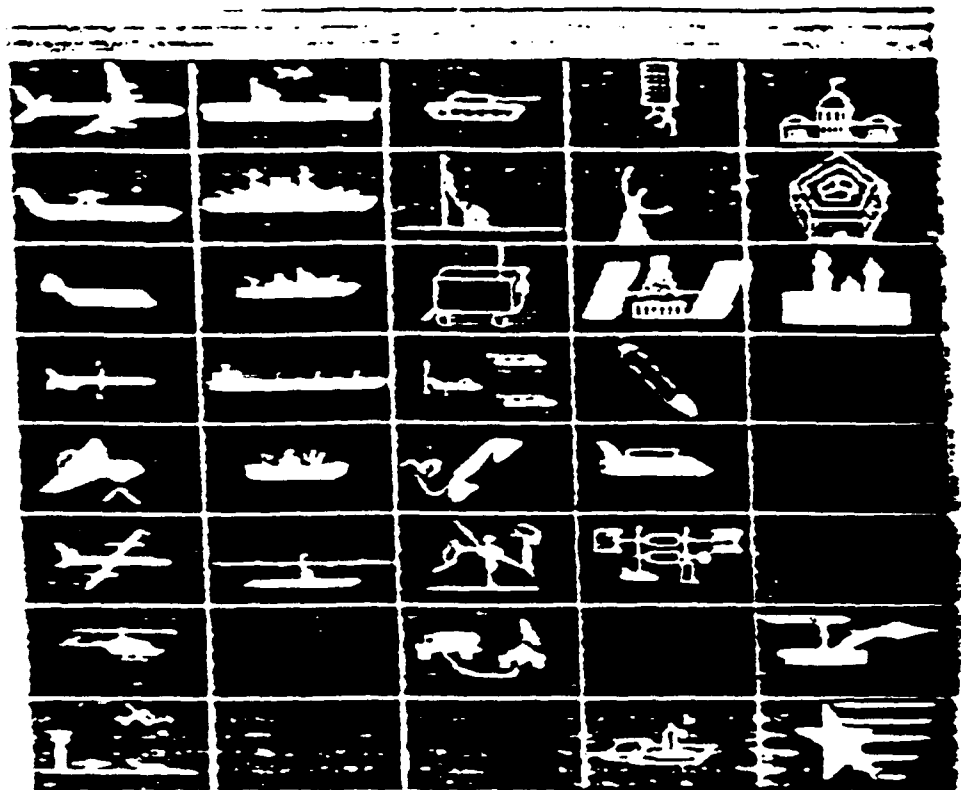


Figure 3. Examples of User-Defined Icons

3.4 Graphical Output

PAWS currently provides graphical output by allowing histograms of various quantities to be displayed, e.g., response times, queue lengths etc., within a single simulation run. It is desirable to provide a more powerful facility to the user, that will allow the graphical output of simulation results across several simulation runs, in terms of line graphs, bar graphs etc. However, it does not seem appropriate for IRA to expend effort in developing the actual graph-drawing software, given the many high quality graph packages commercially available. We propose to store the simulation output in a standard format. A separate program can be written to convert the simulation output to the input format of the user's chosen graph package. IRA proposes to provide such a program for one such environment to the Army, as part of the Phase II effort. Conversion programs for other environments can be developed either by the Army or contracted out.

The benefits are to:

- 1) make the graphical display of statistics from a single run easier,
- 2) store statistics from a single simulation run in a form that is convenient for both machines and humans to read,
- 3) take statistics from several simulation runs and display graphs of key quantities versus changes that occur in model parameters between the simulation runs (or versus simulation number), and
- 4) allow the user to specify the type and detail desired in the graph.

3.5 Network Topology

The user should be able to construct a model that reads in the network topology from a database. It will be useful to provide a facility that will output a network topology graph constructed from a user-specified topology table. Such a graph will be valuable for providing an intuitive feel for the network to be modeled as well as for visually verifying that the topology data is correct.

4.0 Modeling Methodology

Communications systems are typically built up out of standard hardware and software components. To exploit this replication, our approach is to allow a user to build a parametrized submodel representing a certain component of the system, and store it in a library. When needed, a submodel can be extracted from the library and the appropriate parameters filled in by the user. This approach permits rapid construction of performance models, allowing the user to build upon previous work, and makes it easy to answer "what if" types of questions. In addition to the submodels, certain commonly-used algorithms can also be stored, in a parametrized fashion, and incorporated as needed, e.g., routing algorithms for circuit-switched networks.

The following sections deal with enhancements to C/PAWS that will help the model developer to adopt a powerful modeling methodology.

4.1 Submodels and Libraries

We have found that it is difficult to verify the correctness of models of communications systems, largely due to the complexity of the communications systems themselves. To manage this complexity, it is essential to exploit the natural layered aspect of communications systems by developing hierarchical performance models. A step further is to re-use the models for common communications subsystems. The model developer will then have a library of communications submodels that will substantially reduce the modeling effort.

Unlike PAWS 3, the submodeling facilities in C/PAWS will provide convenient parameter-passing facilities similar in power to those for procedure calls in conventional programming languages. This will provide the modularity and information-hiding needed for model reusability.

An important addition to C/PAWS is a versatile and general-purpose library facility, comparable in power to those available for programming languages. A library facility for submodels will be available that:

- 1) allows convenient storage and retrieval of complete submodels of arbitrary size,

- 2) allows a user to specify a library by name as input to a C/PAWS compilation, and
- 3) allows a library to be searched sequentially, in order, so that external references in the main model can be resolved.

Submodels such as our Ethernet model [FER 84] and a token ring model [VEL 86] would be suitable for inclusion in a communications submodel library.

4.2 C Function Library

Certain common aspects of communications systems are better captured using programs written in a high-level language rather than submodels. An example is a set of functions to implement various common routing algorithms for switched communications networks. It will be useful to develop a library of such reusable functions written in C for use by the model developer. The developer may then develop forms or menus for these functions in order to obtain parameters from the model user, just as was done for submodels.

C functions such as those for various routing algorithms, eg. static shortest path and flooding would be suitable for inclusion in a C function library.

4.3 Integrity Constraints and Reasonableness Checks

Model integrity constraints and consistency checks will be important for modeling communications protocols. The following sections present some alternatives in this area.

4.3.1 Node and Transaction State Integrity

The C/PAWS user will be able to insert statements in each node to cause transactions to test the integrity and consistency of transaction, node, and submodel state variables. The C language should serve well here.

4.3.2 Statistics Integrity

The C/PAWS user will also be able to insert statements in the main function to test the integrity and consistency of performance statistics at the end of the simulation. For

example, one might want to check that the sum of the throughputs at nodes A and B equals the throughput at node C.

4.3.3 Submodel Integrity

The C/PAWS user may want to check the consistency of some statistics concerning objects local to some submodel. Assuming path names for library submodels are available, one can embed integrity checks into a model to be placed in a library as follows. For each submodel in the module, write a C function to perform the submodel integrity check and place the C function in the global environment of the module. As part of the external specification for the module, state that each such integrity function should be called by the main function at the end of the simulation.

4.3.4 Reasonableness Checks

Often one has some idea before running a model what the range of possible values are for various performance statistics. For example, one might know that mean response times less than one second or greater than one minute are unreasonable. A mean response time of 5 hours would almost certainly indicate a modeling error. With PAWS 3, the user must search the statistics report manually for unreasonable statistics.

C/PAWS may allow the user to declare the reasonable range of values for each requested statistics. C/PAWS could then highlight in some prominent manner those statistics falling outside these ranges. The user could implement such reasonableness checks in C. However, a declarative language extension would be preferable.

4.4 Statistical Requests

In PAWS 3 statistics requests are embedded within individual nodes and submodels. The submodel developer may not know which statistics the user desires. In C/PAWS this problem will be overcome by a) allowing statistics request statements in the global environment to reference submodels drawn from a submodel library by a full path name or b) allowing conditional statistics requests, which would cause the specified statistics to be collected if a condition (evaluated at RESET time) were true. A library submodel may contain statistics requests for all statistics of interest, which could then be turned on by the user as desired.

A related issue is the ability to separate statistics requests and simulation control from the model specification. This will be addressed in ES/PAWS and C/PAWS by allowing separate compilation of model modules, so that the entire model need not be retranslated when changing a few parameters. User interfaces that obtain parameters interactively at run time will also be useful here.

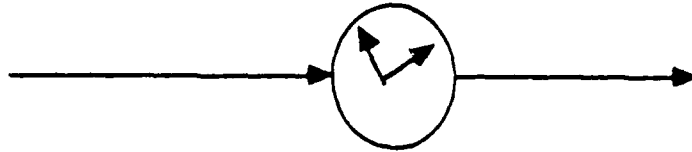
5.0 Communications Systems Features

All the enhancements discussed in this report will make modeling of communications systems easier. In this section we discuss C/PAWS features that will be especially useful for communications modeling but may not be relevant for other applications.

5.1 Timeout Processing

Timeouts are a very widely-used mechanism in communications protocols, particularly at the lower layers of communications systems. The use of timeouts for a data link control protocol has been modeled in this project [IRA 88]. We propose to incorporate a special node type for timeout processing in C/PAWS. The icon and abstract semantics of a TIMEOUT node type are shown in Fig. 4. This figure specifies the actions to be taken by C/PAWS when various events occur at a TIMEOUT node using intuitive pseudo-C statements. The usage of such a timeout node is shown in Fig. 5.

Timeout Node



Semantics (pseudo - C)

```
switch (event) {  
  case NEWMESSAGE_ARRIVED:  
    save original for retransmission  
    save EXPECTED_ACK and MAXREXMIT information  
    set TIMEOUT interval  
    transmit one copy; break;  
  
  case ACK_ARRIVED:  
  
    if this is EXPECTED_ACK for some original msg  
      remove TIMEOUT event  
      mark original message as ACKEDMSG  
    else  
      mark this as a BAD_ACK  
      propagate original and ACK message; break.  
  case TIMEOUT:  
    increment retransmission count  
    if retransmit count > MAXREXMIT  
      mark original msg as TOOMANYREXMIT  
      propagate original message  
    else  
      set TIMEOUT interval  
      transmit one copy  
    break  
  default :  
    error  
}
```

Fig. 4 Timeout Node Semantics

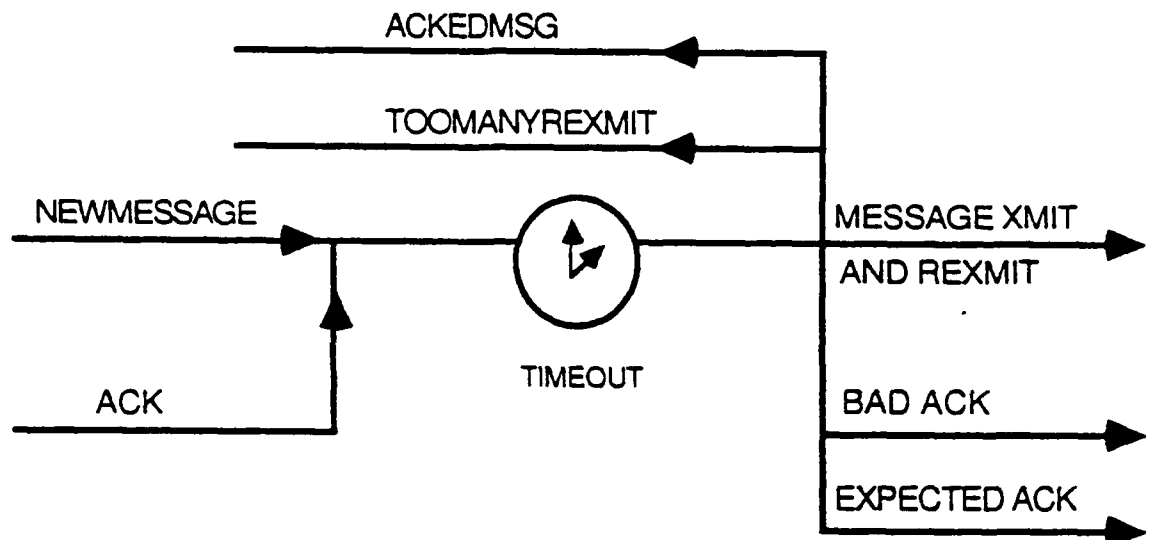


Figure 5. Timeout Node Usage

5.2 Flow Control

Flow control in communication systems is intended to prevent a fast sender of messages from overflowing the capacity of a slow receiver. Two schemes are common. In the first, the sender produces messages up to a fixed predetermined number and waits for permission from the receiver to send any more. In the second, the sender continues to send messages until instructed by the receiver (via a choke packet, XOFF signal, or CTRL-S character) to stop doing so; the sender resumes upon receiving an explicit signal from the receiver.

In C/PAWS and ES/PAWS it will be possible to model both these schemes using two nodes. The first can be modeled using an ALLOCATE and CREATE node pair, while the second can be modeled using a SERVICE node and a SET node. Both these mechanisms are quite general and convenient, so it is not felt necessary to introduce a new node type for modeling flow control.

5.3 Routing Algorithm Utilities

Some routing algorithms involve the creation of routing tables, e.g. static shortest path routing [TAN 81, SCH 87] as implemented in this project [IRA 88]. These involve some fairly sophisticated algorithms that a model user may not wish to compute by hand. Such algorithms may be provided by the model developer in the form of C library functions, possibly with a forms or menu-driven interface for a model user.

Communications systems involve other algorithms ancillary to the main modeling task that would be useful to a model user. These could be handled similarly. An example of such a class of algorithms is discussed in the next section.

5.4 Analytic Bottleneck Analysis

One can rule out many network design or configuration options as infeasible with simple analytic bottleneck analysis. The following example is infeasible regardless of what goes on at BALLOCATE and BRELEASE.

```
ASOURCE --> BALLOCATE --> CSERVICE --> BRELEASE --> DSINK
(expo 1)                               (expo 2)
```

Such analytic bottleneck analysis might involve queueing-theoretic (product form) analysis and ad-hoc protocol analysis. An example of the latter is the simple conservative bottleneck analysis carried out in the Task 3 report for the standard flooding algorithm for a switched network. This kind of bottleneck analysis can be extremely valuable both for verifying that a model is correct and for quickly evaluating design options, and would typically be carried out by a communications engineer.

Some programs for analytic bottleneck analysis could be placed in a C function library as discussed in sec. 4.2. It may be desirable for a model developer to provide a form or menu-driven interface to such functions for the model user.

5.5 Hierarchical simulation

Consider a wide-area network consisting of a collection of computers connected by a packet switching network. One might be able to construct an efficient, detailed model of each computer and an efficient model of the packet switching network. However, an efficient, detailed model of the combination of the network and the individual computers may be infeasible.

Suppose we could simulate an individual computer in an off-line experiment and construct an approximate submodel of the computer based upon this experiment. We could incorporate this approximate submodel into the overall network model. The resulting model could yield accurate performance statistics in a much faster, cheaper simulation than the complete, detailed simulation. This is an example of hierarchical simulation.

The motivation for hierarchical simulation is the possibility of dramatically reducing the time and expense of large communications models. Unfortunately, there is no general

method for hierarchical simulation that has been validated. We propose a new method that we expected to work well, but some validation work must be done (in Phase II).

5.5.1 Definition

Hierarchical simulation refers to the following:

- a) separate off-line simulation of one or more submodels,
- b) constructing simplified approximations of these submodels based upon information (e.g. performance statistics) collected in the off-line experiments, and
- c) incorporating these simplified approximations into a higher-level model.

5.5.2 General Approach

The general approach we suggest is to:

- a) allow the user to specify the form of a function mapping submodel and transaction state to transaction response time,
- b) conduct a number of off-line simulations of the submodel, varying submodel parameters in each simulation,
- c) in each off-line simulation, for each transaction, record the model state, transaction state, and response time,
- d) use regression analysis on this data to determine the parameters of the function specified in (a) above,
- e) construct an approximate representation of the submodel that evaluates the submodel and transaction state on entry to the submodel and evaluates the function constructed in (d) to obtain a submodel response time for the transaction.

5.5.3 A simple example

Suppose we specified the following submodel response time function:

$$R = a * N$$

where

R = transaction submodel response time,

N = number of transactions in the submodel, and

a = a parameter to be determined by regression analysis.

We would conduct a number of simulations of the submodel. As each transaction entered the submodel, we would record N , the number of transactions in the submodel. As each transaction left the submodel, we would record R , the response time. By the end of the experiments we would have recorded a large number of (N,R) pairs. We would perform regression analysis to determine the best value of a . We would then construct the approximate submodel:

ENTER -----> DELAY (delay time = $a * N$) -----> RETURN

6.0 Language Enhancements

The model developer using C/PAWS will have a very powerful language available, combining the flexibility and expressiveness of C with the high-level modeling features of PAWS, ES/PAWS, and GPSM. This power will be further enhanced by some of the facilities discussed in this section, such as interrupt resumption nodes and more flexible FORK/JOIN constructs.

6.1 Failure Modeling Methodology

Failure modeling has been studied extensively at IRA [FER 84, PAL 85, VEL 86] and by IRA clients [CON 86], as well as in the context of this project [IRA 88]. The facilities available for failure modeling are flexible and powerful. Due to the special relevance of failure modeling for the Army's communications systems, this aspect of PAWS will be enhanced in C/PAWS as discussed below.

Currently, failure modeling in PAWS involves creation of a failure transaction that interrupts the processing of normal transactions for a time equal to the time to repair the failed component. In C/PAWS this will be made easier using an interrupt resume node described in sec. 6.2. The following failure modeling methodology is then recommended:

- Define a globally visible failure flag for each device. The flag will be true if and only if the device has failed and has not yet recovered.

When a device fails, set its failure flag true and interrupt all transactions in the submodel representing the device.

Use an interrupt resumption node (see sec. 6.2) in that submodel. Typically, interrupted transactions should be destroyed at a sink immediately following the resumption nodes, but occasionally some special processing will be needed.

In the interrupted submodel, place an interrupt edge from the enter node to a sink and from each call node to a sink. Label these edges with the device failure flag. All transactions entering or returning to a failed submodel will immediately sink.

Timeouts specified external to the submodel will cause retransmission of messages, as in the real system.

When a device recovers, set its failure flag to false.

In many cases it will be possible to simplify this modeling approach. It is possible to view the occurrence of a failure as consisting of several phases. Each phase can be modeled independently, under certain assumptions. Thus instead of generating failures dynamically, it is possible to model the behaviour of a system at some phase of its failure mode. This results in shorter simulation runs and simpler models. This methodology is described in Sec. 4.6 of the Modeling and Experimental Evaluation report.

6.2 Interrupt Resumption Nodes

C/PAWS will allow transaction interruptions to be specified more conveniently than PAWS 3. For example, all the transactions in category C in submodel S can be interrupted.

A new interrupt resumption node type will address this issue nicely. There can be at most one interrupt resumption node per submodel. If one is present, all transactions interrupted in that submodel will immediately proceed to the resumption node. If one is not present, interrupted transactions behave as in PAWS 3; they depart their current node along an edge.

Interrupt resumption nodes have no specification. Transactions arriving at such nodes depart immediately along an edge. Interrupt processing now can be specified easily using all the ES/PAWS capabilities.

6.3 FORK and JOIN Constructs

The PAWS FORK and JOIN constructs will be made more flexible in C/PAWS. These enhancements will allow the construction of reusable submodels and the collection of arbitrary statistics easier. Some specific proposed enhancements are:

- 1) Allow a parent to proceed directly through a FORK without delay. The parent will retain its resources instead of passing them to the eldest child. This enhancement will allow parents to participate in the model at the same time as the children.
- 2) A consequence of the first enhancement is to allow a parent to proceed to a JOIN and wait there until all its children JOIN.
- 3) Allow a child to force the release of a parent even before all children have reached a JOIN. This may be useful in communications systems where, once one copy of a message has reached the destination, the original may be destroyed.

6.4 Source Node Control

The model developer sometimes wants a source node to remain inactive for a while at the beginning of the simulation or to halt before the simulation ends. In these cases, one would like to specify start and stop times for the source node. A source node with start time = 500.0 would not begin generating transactions until time 500.0. A source node with stop time = 1000.0 would stop generating transactions at time 1000.0.

Stop times could be used to stop the generation of new messages, and allow the system to consume the messages already generated ("drain" them) before stopping the simulation. A less declarative but more general approach to this problem would be to allow set nodes to affect source nodes. A power of zero would halt a source, a power of one would restore normal behavior. Other positive powers would simply scale the interarrival times.

7.0 Conclusions

A wide variety of enhancements have been discussed to the already powerful features of PAWS 3 in order to make modeling of Army communications systems easier. Although such modeling is already feasible in PAWS, it will become even faster and more efficient in C/PAWS when the enhancements discussed here are incorporated.

An attempt has been made in this report to discuss the enhancements proposed for C/PAWS at a fairly high level. It is felt that specific implementation decisions should be made in the early stages of the Phase II implementation project. This allows CECOM to discuss and review the proposed enhancements, and avoids placing unnecessary constraints on the C/PAWS implementation at this stage of the project.

We believe that C/PAWS will be a highly productive, elegant tool for modeling Army communications systems when the proposed enhancements are incorporated. It is quite feasible to implement these enhancements using a variety of implementation strategies discussed in this report, and in fact some are currently being implemented as part of the ES/PAWS project.

References

BRO 88

J. C. Browne, P. Jain, D. M. Neuse and M. Esslinger, "ES/PAWS - A System Level Design Aid", submitted for publication in *Proceedings of the Design Automation Conference*, June, 1988.

CON 86

Customer confidential document. The model involves a Sperry 1100/44 computer system with multiple command, arithmetic and peripheral processors used for real-time computations. Permission is being sought to disclose this information.

FER 84

V. Fernandes, J.C. Browne, D. Neuse, and R. Velpuri, "Some Performance Models of Distributed Systems", *Proceedings of the CMG XV International Conference*, Dec., 1984.

IRA 87a

Information Research Associates, *Performance Analyst's Workbench System (PAWS) User's Manual*, 1987.

IRA 87b

Information Research Associates *Graphical Programming of Simulation Models (GPSM) User's Manual*, 1987.

IRA 88

Information Research Associates, *Modeling and Experimental Evaluation Report*, Task 3 of Army SBIR Phase I project, draft submitted to CECOM, March 18, 1988.

NAV 87

United States Navy Contract No. N60021-86-C-0145, High Level Simulation of Electronic Systems, 1987.

PAL 85

Annette Palmer, J. C. Browne, J. Silverman, A. Tripathi, and R. Velpuri, "A Performance Model of a Fault-Tolerant Distributed System for Evaluating Reliability Mechanisms", *Proceedings of the CMG XVI International Conference*, 1985.

SCH 87

Mischa Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*, Addison-Wesley, May 1987.

TAN 81

Andrew Tanenbaum, *Computer Networks*, Prentice-Hall, 1981.

VEL 86

Rajkumar Velpuri, "Performance Study of Zeus Distributed System with Different Communication Networks", *M.S. Thesis*, The University of Texas at Austin, May, 1986.

Appendix C

Outline of PAWS and GPSM

Introduction to Performance Analyst's Workbench System (PAWS) and Graphical Programming of simulation Models (GPSM)

1.0 Introduction

GPSM is a graphical modeling language that takes advantage of the natural human affinity for pictorial and visual presentation of information. In the past, simulation models have been coded in a variety of general purpose procedural languages such as FORTRAN and PASCAL. More recently, a number of higher level languages specifically designed for simulation modeling have made it possible for the modeler to concentrate more on the details of producing an accurate model and less on the details of coding that model.

One of the more powerful and successful examples of such simulation languages is the Performance Analyst's Workbench System, PAWS. PAWS provides the modeler with a number of high-level primitives such as a variety of queueing disciplines (first-come-first-served, first-fit, priority, etc.), probability distributions (uniform, exponential, ERLANG, etc.), and output statistics (throughput, queue lengths, queueing times, etc.). PAWS also uses a pictorial representation of abstract queueing networks called Information Processing Graphs (IPGs) as both a design tool and a documentation method for the simulation models that are ultimately coded in the highly declarative PAWS language. These IPGs have tended to serve a purpose in the world of simulation modeling analogous to the use of flow charts and other visual aids in general-purpose procedural programming. This despite the fact that queueing networks seem to have a much more natural graphical representation.

Very recently, the availability of low-cost, medium- and high-resolution graphics machines has made it feasible to use the techniques of graphical programming to produce simulation models directly. The GPSM system is a tool that allows IPGs to be drawn and modified directly on the graphics screen of any IBM-PC compatible machine using a mouse as a pointing device. Such graphs, though still very useful for design and documentation purposes, may be automatically translated into simulation programs in the PAWS language. This enables modelers to deal directly and exclusively with the pictorial information in the IPGs in order to design, execute, and refine their simulations.

By using GPSM as a graphical interface to PAWS, all the advantages of a pictorial programming language may be obtained without sacrificing the power and versatility of a compiled, specialized simulation language. The visual nature of the graphical interface greatly speeds and simplifies the transfer of ideas into symbols by utilizing the most natural and straightforward kinds of symbols: pictures.

A performance modeling project using GPSM and PAWS might proceed as diagrammed in Figure 1.

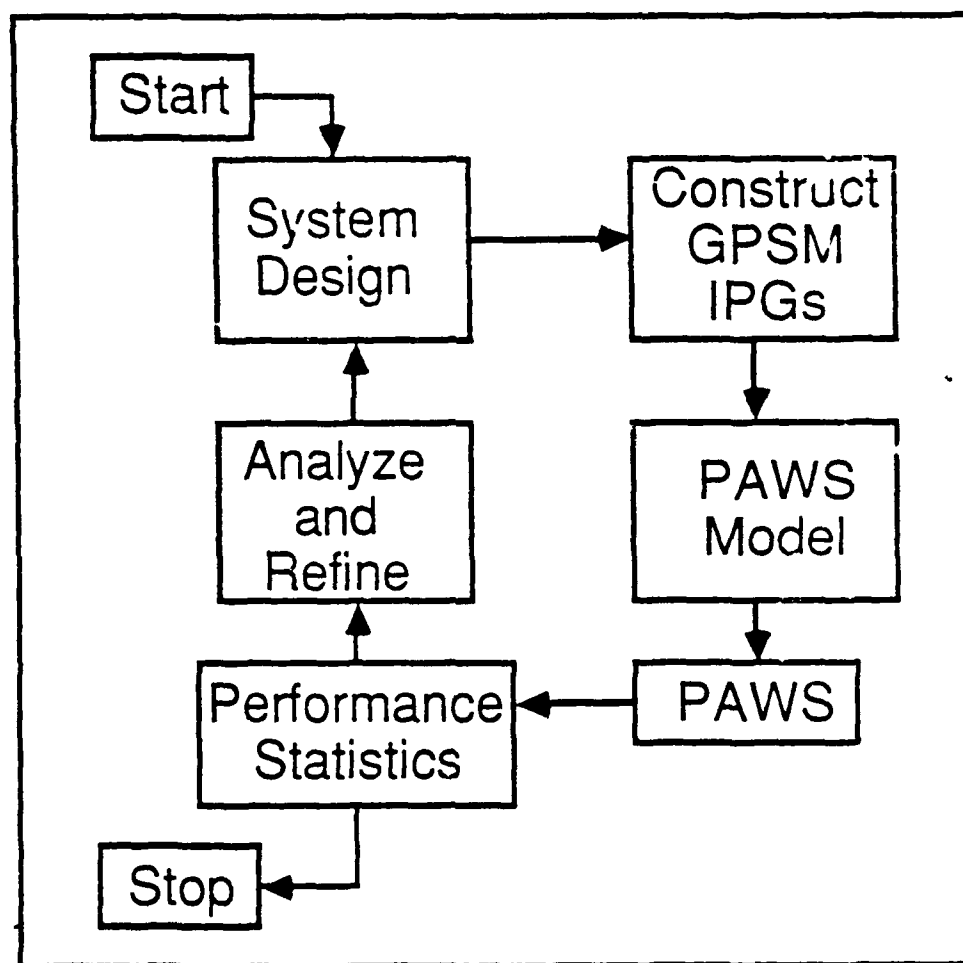


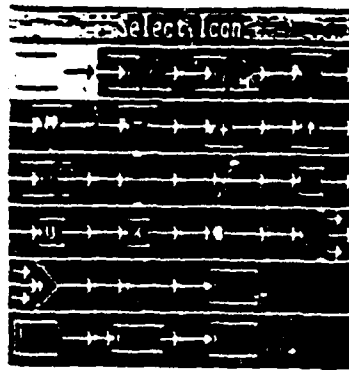
Fig. 1

An analyst will generally abstract information from the initial system design in order to draw information processing graphs (using GPSM) that capture the important performance characteristics of the design. Further information such as service distributions and

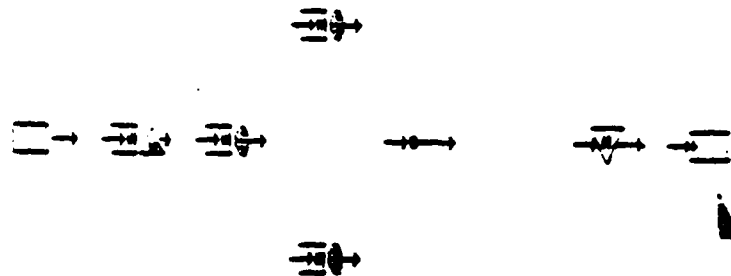
queueing disciplines for the active queues are then added to the graphs as attributes of the queues themselves. When all of the required information has been added to the graphs, GPSM will then be used to translate the pictorial form of the simulation model into the declarative, textual form expected by the PAWS simulation language. This completely automatic translation results in a simulation program that can be compiled and executed with PAWS in order to obtain statistical data and performance estimates of the system design. Upon examining the resulting estimates, the analyst may choose to refine the system design, modify the graphical representation of the model maintained by GPSM, and repeat the cycle until the observed performance estimates obtained from PAWS meet the desired (or required) performance.

2.0 Example GPSM Graph

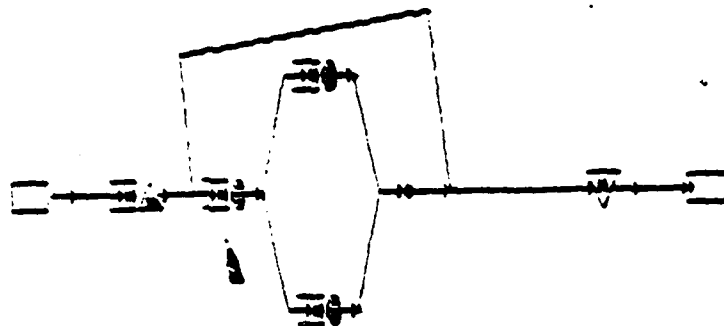
The following sequence of pictures illustrates the construction and translation of a simple GPSM simulation graph. The example we will use is a very simple model of a computer system including memory, CPU, and two disk devices. In the interest of brevity we have modeled each of the disk subsystems as a single service node. The pictures are a series of snapshots of the GPSM graphics screen at various points during a typical interactive session.



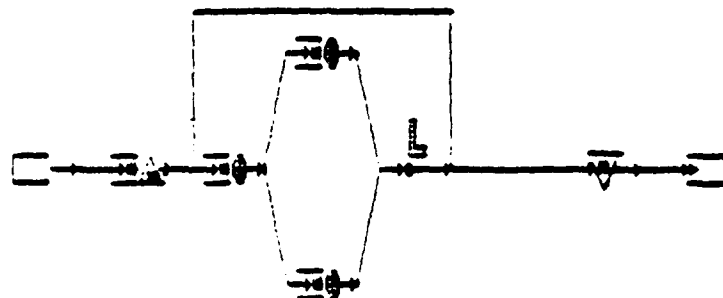
Selecting a Node Icon



Placing the Node Icons



Connecting the Nodes with Arcs



Squaring the Arcs

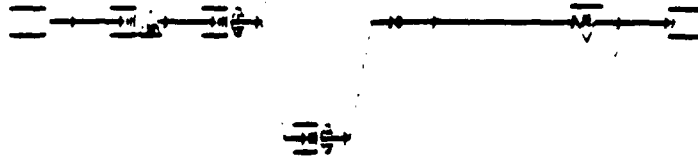
Node Name
 allocate
 Description
 Allocate main memory
 specification fl-weight
 quantity 500 mainmem firstfit;
 qd cfs
 request (batch.all) uniform (10,50)
 mainmem 11(3);

1

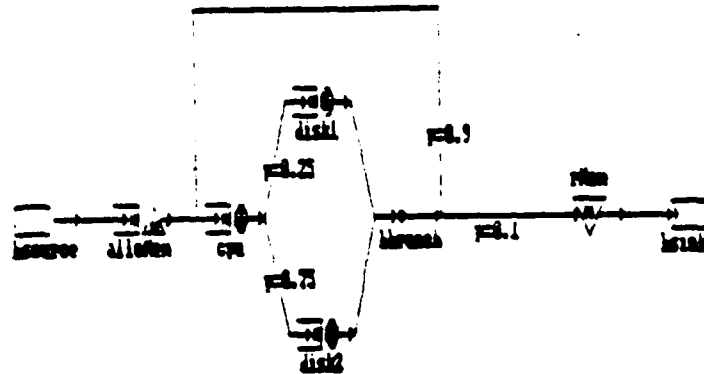
Opening the Memory Allocation Node

Arc Label
 p=0.25
 specification fl-weight
 (batch.all) 0.25;

2



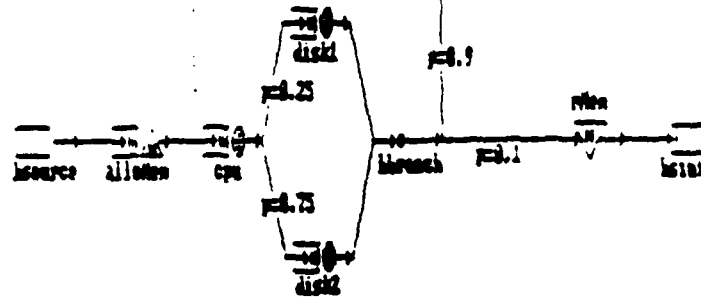
Opening an Arc Definition



The Completed Graph with Labels Displayed

Translate

Exit



Translating the Graph to PAWS

3. GPSM Node Icon Summary

GPSM Icons

